12-2017

Accelerating Stochastic Random Projection Neural Networks

Swathika Ramakrishnan
sxr1661@rit.edu

Accelerating Stochastic Random Projection Neural Networks

Swathika Ramakrishnan

Accelerating Stochastic Random Projection Neural Networks

by

Swathika Ramakrishnan

December 16, 2017

A Thesis Submitted
in Partial Fulfillment
of the Requirements for the Degree of
Master of Science
in
Computer Engineering

Supervised by

Dr. Dhireesha Kudithipudi
Department of Computer Engineering
Kate Gleason College of Engineering
Rochester Institte of Technology
Rochester, New York
December 2017

R·I·T | Kate Gleason *College of* ENGINEERING

Department of Computer Engineering

Accelerating Stochastic Random Projection Neural Networks

Swathika Ramakrishnan

Committee Approval:

Dr. Dhireesha Kudithipudi — Date
Thesis Advisor, Department of Computer Engineering

Dr. Marcin Lukowiak — Date
Committee Member, Department of Computer Engineering

Dr. Sonia Lopez Alarcon — Date
Committee Member, Department of Computer Engineering

Acknowledgments

I would like to thank my thesis adviser Dr. Dhireesha Kudithipudi for her constant support and guidance throughout my thesis.

I would like to thank Dr. Marcin Lukowiak and Dr. Sonia Lopez for serving in the committee and for their valuable time.

Special thanks to Dan Christiani, past member of the NanoComputing lab, for his immense patience and guidance in this field.

To the members of the NanoComputing Lab for the technical discussions

Thanks to the RIT CE deptartment for the resources and support.

To my parents for their trust in me

Abstract

Accelerating Stochastic Random Projection Neural Networks

Swathika Ramakrishnan

Supervising Professor: Dr. Dhireesha Kudithipudi

Artificial Neural Network (ANN), a computational model based on the biological neural networks, has a recent resurgence in machine intelligence with breakthrough results in pattern recognition, speech recognition, and mapping. This has led to a growing interest in designing dedicated hardware substrates for ANNs with a goal of achieving energy efficiency, high network connectivity and better computational capabilities that are typically not optimized in software ANN stack. Using stochastic computing is a natural choice to reduce the total system energy, where a signal is expressed through the statistical distribution of the logical values as a random bit stream. Generally, the accuracy of these systems is correlated with the stochastic bit stream length and requires long compute times.

In this work, a framework is proposed to accelerate the long compute times in stochastic ANNs. A GPU acceleration framework has been developed to validate two random projection networks to test the efficacy of these networks prior to custom hardware design. The networks are stochastic extreme learning machine, a supervised feed-forward neural network and stochastic echo state network, a recurrent neural network with online learning. The framework also provisions identifying optimal values for various network parameters like learning rate, number of hidden layers and stochastic number length. The proposed stochastic extreme learning machine design is validated for two standardized datasets, MNIST dataset and orthopedic dataset. The proposed stochastic echo state network is validated on the time series EEG dataset. The CPU models were developed for each of these networks to calculate the relative performance boost. The design knobs for performance boost include stochastic bit stream generation, activation function, reservoir layer and training unit

of the networks. Proposed stochastic extreme learning machine and stochastic echo state network achieved a performance boost of 60.61x for Orthopedic dataset and 42.03x for EEG dataset with 2^{12} bit stream length when tested on an Nvidia GeForce 1050 Ti.

Contents

List of Figures

List of Tables

Acronyms

2D

two-dimensional

3D

three-dimensional

ANN

Artificial Neural Network

BSL

Bipolar Stochastic Logic

ELM

Extreme Learning Machine

ESN

Echo State Network

MAE

Mean Absolute Error

MSE

Mean Square Error

NC

Neuromorphic Computing

RNN

Recurrent Neural Network

SC

Stochastic Computing

SNL

Stochastic Number Length

Chapter 1

Introduction

1.1 Motivation

The term Neuromorphic Computing (NC) was first introduced in 1980s by Carver Mead for the electronic system that can compute similar to the nervous system. Over the years the term NC has evolved to represent more concepts thus bridging the gap between neural and computer systems [4]. NC has outperformed traditional systems with its striking improvement in the last two decades in the field of image processing, speech recognition, etc, where the traditional systems have failed notably to achieve a good performance. NC being million times more power efficient than Von Neumann systems, can lead to a great leap in improving the computational capabilities [5].

Major differences between Von Neumann systems and NC are based on the organization of memory and the processing method. There is a separate memory and processing unit in Von Neumann systems between which the information is shuttled multiple times to carryout the process whereas in NC there is a distribution of memory with the processing unit thus saving the power spent on additional cores used in a digital system [6]. Other advantages of neuromorphic systems are their ability to self learn, perform complex parallel processing and robustness in adapting to the error prone environment making it one of the important topics to be researched on.

In the field of neuromorphic systems, Artificial Neural Network (ANN) has been

widely used in various applications like gaming, speech recognition, mapping, etc. In recent years there has been an immense interest towards the hardware implementation of these network to achieve distributed memory, high network connectivity and better computational capabilities that are lacking in software implementation of the same [7]. A custom hardware design can work much faster with low power [7].

With the ever increasing need for more reliable, robust and a small footprint, stochastic logic can be used in the implementation of ANNs to provide an highly error tolerant system. Stochastic Computing was first introduced as a solution to challenges faced in the implementation of machine learning systems as it has the ability to store bits for long period, increment in small steps, has simpler arithmetic operations and high error tolerance [8]. Thus a stochastic implementation of an ANN, where multiplication, summation and activation are the three major steps, can be performed by simple circuits like AND gate, MUX operation and thresholding respectively. Stochastic logic involves computation on a long stream of random bits. In order to achieve an acceptable accuracy, the length of the bit stream needs to be increased thus leading to a long computation time for processing these stochastic bits [9].

The operation on these bits streams are bitwise and have no inter dependencies. Exploiting this feature, the stochastic computation process can be parallelized and accelerated in GPU to reduce the computation time. For example, multiplication of two n-bit stochastic numbers takes 'n' steps in CPU, where as in GPU it can be implemented in one parallel operation. When using stochastic logic in ANN, which has no inter dependencies within the layers, can be accelerated layer by layer. This way Large and different types of Stochastic ANNs can be accelerated in GPU to aggressively test and validate the network to arrive at the optimal architecture before custom designing in hardware.

1.2 Research statement

Goal of this thesis to develop a GPU accelerator for a stochastic implementation of two types of ANN : Extreme Learning machine (ELM), a feed-forward neural network and Echo State Network (ESN), a Recurrent Neural Network (RNN) to provide an accelerated platform for testing stochastic implementation of ANN by addressing one of its major drawbacks, long computation time required to process the stochastic bits. This will provide as a foundation for the hardware implementation of the same. Following steps where taken to accomplish the thesis goal

1. Develop GPU acceleration framework to validate a stochastic logic neural networks at scale

2. Identify the design knobs for performance boost - stochastic bit stream generation, activation function, and training unit of the network

3. Validate stochastic neural network designs on standardized datasets

4. Create an exact CPU model to compare the performance with the GPU models

The rest of this document is structured in the following manner: First chapter explains the motivation behind this work, research statement and the background for the history of stochastic computing, introduction to stochastic numbers, ANN and accelerating the network in GPU. The second chapter talks about the Design methodology for the proposed work and the building blocks. Chapter 3 explains the layer by layer acceleration of stochastic ELM in GPU. Chapter 4 the GPU implementation of ESN focusing on the reservoir layer in particular. Chapter 5 explains the the Data sets used in detail. Chapter 6 contains the results and analysis for ELM and ESN. Chapter 7 consists of the conclusions, discussion and future study.

1.3 Background

1.3.1 History of Stochastic computing

In 1953, stochastic computing (SC) was first introduced by John Von Neumann [10]. Due to the lack of advancements in computing till 1960s, SC was not fully developed. In 1965, SC emerged as a low cost computing technique to replace the traditional binary systems [9]. This was mainly developed as a part of a research to make advancements in realization of automatic controllers in the field of machine learning and pattern recognition [8].

Conventional digital systems can be used to perform identification and search operations but will not help in the construction of complex computing structure required for machine learning [11]. This structure involves in the design of storage elements which can store bits for a long time and increment values in small steps [11]. SC can perform the above operations much better compared to a traditional system, which will require more elements comparatively. SC being probabilistic in nature can increment in small steps and also the basic operations on stochastic numbers can be performed using the simple circuits like AND gate for multiplication and 2:1 MUX for addition. SC is also a highly error tolerant system because the position of the bits in the stream doesn't matter but only the count of 1's. Flip of a single bit in a stochastic bit stream will not significantly affect the accuracy compared to a conventional system where a flip of a bit can be catastrophic [9].

Many such machines were developed in the year 1969 to 1974 [9]. Despite the growth in SC, conventional digital system still had its advantages over SC due to the following reasons. Despite the low accuracy of the system due to its random nature and the requirement for long stochastic bits streams to attain a acceptable accuracy it has been gaining attention past few years due to its less complex arithmetic units and error tolerance. It is a good fit for the current technologies which requires a

highly reliable, low power and small systems [9]. When it comes to certain fields like pattern recognition and machine learning, the advantages of probabilistic nature of SC can outrun the traditional system.

1.3.2 Introduction to Stochastic Numbers

For a long time analog and digital were the fundamental number representations. Later the stochastic numbers were introduced. In stochastic logic a signal is conveyed through the statistical distribution of the logical values and is represented by a random bit stream. The fractional numbers correspond to the probability of occurrence of a logical one versus a logical zero[8]. Mathematically, the stochastic numbers are represented by a bit-stream X with N bit length. For example: if 60% of the bits are 1s and the rest are 0s, then the value is 0.6 and can be represented by a equivalent stochastic bit stream, 1001101110. Where only the count of the 1s are taken into account and not the position of the 1s in the bit stream. While converting the floating point number to stochastic bit stream the precision of the number scales down. For example: to convert a 2^5 bit precision floating point number, stochastic bit stream length of atleast 10^5 is needed. To convert it back to decimal number, sum of the 1's in the bit stream by the total number of bits in the stochastic bit stream is performed. There are two types of stochastic number representations [8], single line unipolar and single line bipolar representations. Throughout the work bipolar stochastic numbers are used for representing the stochastic bit streams as the ANN requires symmetric encoding.

Single Line Unipolar Representation

It is the most basic form of representation for values between interval [0,1]. Let the length of the bit stream be N and the probability that any bit in the N bit stream will be 1 is given by P1. Count of 1s in the bit stream is given by E. Then the stochastic

number in single line unipolar format is given by the equation 1.1.

$$P_1 = \frac{E}{N} \tag{1.1}$$

$$P_1 = \frac{4}{10} = 0.4; Bitstream => 0101000011 \tag{1.2}$$

From example 1.2, Where 10 is the total number of bits (N). N will be denoted as stochastic number length (SNL) in this thesis and 4 is the total number of 1s in the stochastic bit stream then the stochastic bit. Two unipolar stochastic numbers can be multiplied using an AND gate and the addition can be performed using a 2:1 MUX.

Single Line Bipolar Representation

This format is used when there is a need to represent numbers from -1 to 1. Let the stochastic bit stream length be N, let E be the count of number of 1s in the bit stream and P2 is the probability of 1 in the bit stream.

$$P_1 = \frac{E}{2N} + \frac{1}{2} \tag{1.3}$$

$$\frac{E}{N} = 0.5 = \frac{4}{8}; P2 = \frac{3}{4} = \frac{6}{8} => 01101111 \tag{1.4}$$

$$\frac{E}{N} = -0.2 = \frac{-1}{4}; P2 = \frac{3}{8} => 10011000 \tag{1.5}$$

Stochastic Number length required for an equivalent precision is 2N. This method is used in this thesis for representing the stochastic number. Two stochastic numbers in bipolar formatted can be multiplied using an XNOR gate and summed using a 2:1 MUX. In a stochastic system both unipolar and bipolar can coexist. While generating

stochastic bit streams, there might be correlation among numbers and the number can be altered by physical errors. This can be rectified by using a suitable stochastic number generator [9].

1.3.3 Artificial Neural Network

ANN is a learning system inspired by a biological neural network, where the input data is used to train the algorithm to map the input to the corresponding output. The processing units are called neurons, the connection between the neurons are called synapse. Each of these edges have a weight value to indicate the strength of the connection [12].

Key advantages of ANN over the traditional Von Neumann architecture are its ability to learn and model the complex non-linear relationships between the inputs and outputs, it can generalize and predict unseen relationships by learning the initial inputs and there are no limitations on the input variables [13].

There are two types of ANN, feed-forward neural network (FNN) and recurrent neural network (RNN). In the first type, the input is mapped to output by passing the input through the layers of the network without any feedback as shown in Figure 1.1, which is a one hidden layer FNN. The hidden layer output 'H_1' is calculated as shown in equation 1.6, where 'W_{in1}' is the synaptic connection between the input X and hidden node H1, b is the bias value and g(x) is the activation function. In RNN the output is fed back and depends on the previous state calculations that are captured as shown in Figure 1.2 [14], where the reservoir node 'R_1' is calculated using the equation 1.7, where 'W_{in1}' is the synaptic connection between the input and 'R_1' and 'W_{x1}' is the synaptic connection between the reservoir node 'R_2' and 'R_1'.

$$H_1 = g(X.W_{in1} + b) \tag{1.6}$$

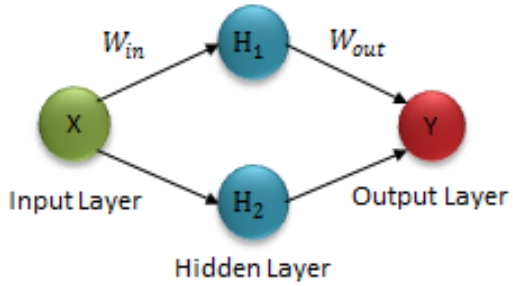

Figure 1.1: Example feed forward neural network with one input, hidden and output layer

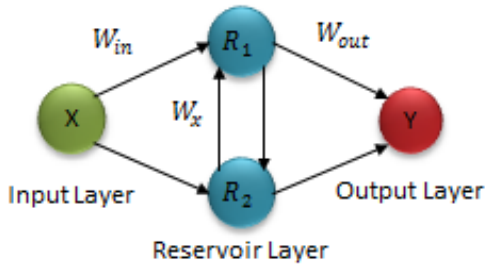

Figure 1.2: Example RNN with one input, reservoir and output layer

$$R_1(n+1) = g(X.W_{in1} + R_1(n).W_{x1} + b) \tag{1.7}$$

1.3.3.1 Extreme Learning Machines

ELM was introduced in the year 2005 and proved to be faster than the traditional gradient based learning algorithm and less human intervention [15]. ELM is a feed-forward neural network consisting of a single hidden layer, the input is mapped to output by passing the input through the layers of the network without any feedback as shown in Figure 1.3. Input weights between input layer and hidden layer are initialized randomly. The hidden weights between the hidden layer and output layer are initialized randomly at the start and then updated at every epoch to arrive to the final weight values. The input weights remain unchanged throughout the process unlike multilayer perceptron network (MLP), where the error is diffused backwards in the network to update all the weights. Number of nodes in the input layer depends on the number of features and the hidden layer acts as an abstraction layer by making the inputs linearly independent of one another by projecting it to a higher dimensional space [15].

There are two types of learning: batch learning and online learning. In this work, on line learning will be used. Let us assume there are X input layer nodes, number

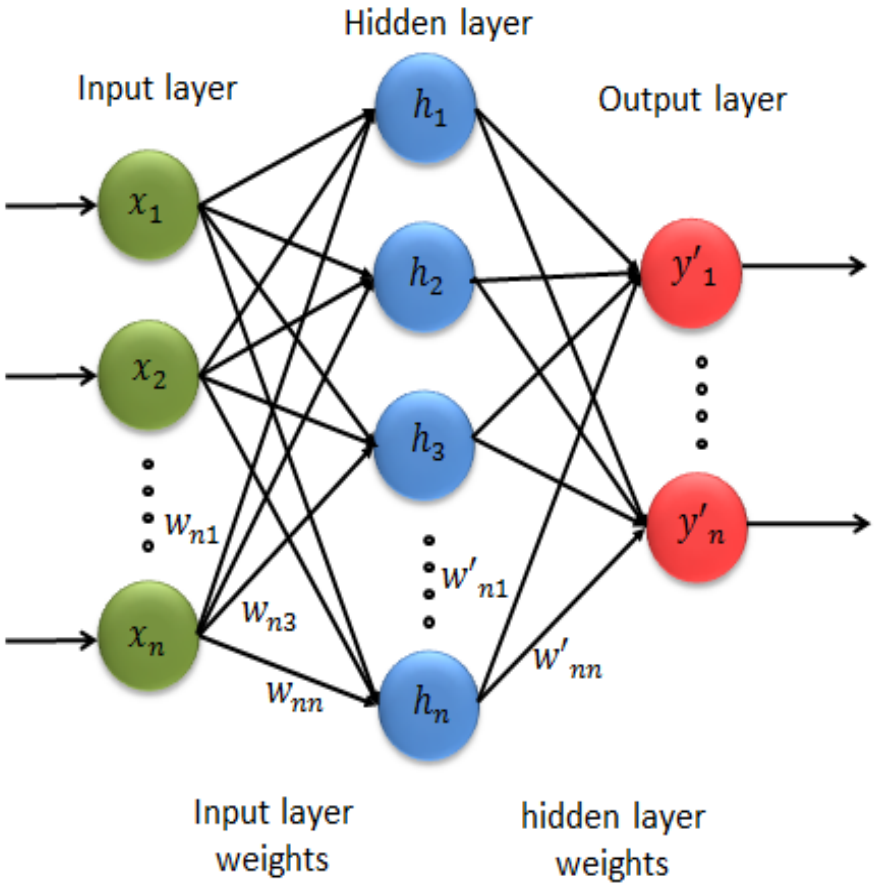

Figure 1.3: Architecture of ELM with one input, hidden and output layer.

of samples be N and the activation function is denoted by g(x). Let H be the matrix containing the output of hidden layer nodes, which is given by equation 1.8.

$$H = h_{ij} = g(W_j.X_i + bias_j), 1 \geq i \leq N, 1 \geq j \geq N \tag{1.8}$$

where the product of weight (W_j) and input matrix (X_i) is summed with the bias value before feeding into the activation function. the output of the activation function is represented by h_{ij}. Output of the network is presented by Y_i and the target output is $Y1_i$. Sigmoid-like activation function is used in this work for the activation of the stochastic bit streams as shown in equation 1.9 [16].

$$\Delta W = \alpha(Y1_i - Y_i)h_ij \tag{1.9}$$

$$W2 = W2 - \Delta W \tag{1.10}$$

$$g(x) = \frac{1}{1 + e^{-x}} \tag{1.11}$$

1.3.3.2 Echo State Network

ESN is a RNN introduced by Harbert Jaegar in the year 2010 [17]. They are a partially trained neural networks used for spatiotemporal signal processing like speech recognition, bio signal processing, etc. [18]. It consists of a input layer, reservoir layer and an output layer as shown in Figure 1.2. Reservoir layer consists of recurrent connections which help in extracting desired features within a time window. The connections from input to reservoir layer and inside the reservoir layer consist of randomly generated weights, which are not updated when the network is trained. These two layers help in extracting the input signal's temporal data. Only the weights connecting the reservoir layer to the output layer are updated over every epoch. Therefore the training of ESN is less complicated and fast compared to other RNNs

The Main goal of the network is to update the output layer weights with the response from the reservoir layer. First step is to calculate the reservoir layer output using equation 1.11, where, X[n] is the input signal to the network, H[n] is the output of the reservoir layer, W_{in} is the input layer weight matrix, W_x is the reservoir layer weight matrix, W_{out} is the output layer weight matrix and the reservoir layer output for the next input signal is H[n+1], g(x) is the reservoir layer activation function shown in equation 1.10. There are several methods to update the output layer weights. This work uses a linear regression method to calculate the output layer weights and the equation used is same as the ELM as shown in equation 1.8 and 1.9.

$$H[n+1] = g(W_in.X[n+1] + W_x.X[n]) \tag{1.12}$$

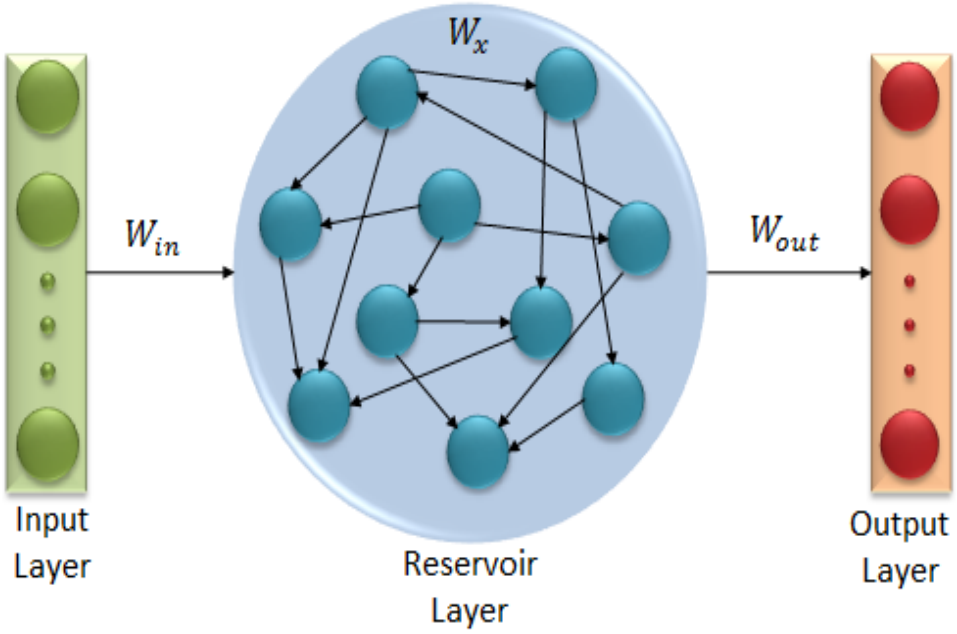

Figure 1.4: Structure of echo State Network with one input, reservoir and output layer

1.3.4 Accelerating neural networks with GPUs

A graphical processing unit (GPU), is an electronic circuit that can manipulate in parallel to accelerate the speed of an operation. It is mainly used in the field of image processing, gaming, embedded systems, etc [19]. They are relatively new, complex and delivers high theoretical computation power for a comparatively low cost [20]. It has overtaken general purpose CPUs in various fields like scientific computation, graphics and image processing by its high processing speed, highly parallel structure and programmability. [21].

Complex and large computation of ANN is one of the major challenges in using it for various applications. The ANN process can be accelerated by the use of GPU much faster than in a CPU for the following two main reasons (i) GPU has a fatser bandwidth and large number of resourses (ii) neural network computation which majorly involves performing similar processes on multiple inputs will be a good fit to GPU architecture [22]. There is no special hardware developed exclusively for an

ANN but the available resources can be adapted for the implementation of different application [22]. These processors are quiet different from the general CPUs as they are generally mounted on the main memory's internal board through a PCI-E connecter, thus it has its own large memory as the central memory. It copes with massively parallel problems with its large number of processing cores compared to powerful computing units which are small in number.

Compute unified device architecture (CUDA) is a framework designed to program directly on a GPU. This framework is composed of C-like language with extended keywords to program in GPU and CPU [20]. Especially a stochastic implementation of an ANN has no inter dependencies and the operations on the same is performed bitwise, thus making it convenient for parallelizing the whole operation and running the network on GPU. For example: multiplication of two stochastic bit streams are performed bitwise using a 2-input XNOR gate, these bits are processed in a sequence one after the other in CPU, whereas in GPU all these operations can be performed in one parallel step. Similar works include the GPU implementation of multilayer perceptron (MLP) [22], implementation of GMDH neural network .[23] and GPU implementation of spiking neural network implementation for a real - time systems [24].

Nvidia with the CUDA architecture follows the binary floating point arithmetic with IEEE standard introduced in the year 1985 [25].To achieve highest performance, it is important to know all the aspects of floating point behaviors to implement correct numerical algorithms [26]. With each new hardware generation, Nvidia has extended the GPU's capability. Both the single and double precision is supported by current generation GPUs [20]. In Nvidia GPUs each floating point instruction has an encoding rounding mode where as a floating point control word is used in x86 architectures. There is no trap handler or status flag in GPU for indicating inexact arithmetic calculation. Thus there will be some accuracy drop in GPU compared

CPU [26].

1.3.5 Summary

Stochastic implementation of ANN significantly reduces the area and power with a simple arithmetic unit and are highly error tolerant but there occurs an accuracy drop due to the random nature of these systems which can be reduced by using long stochastic bit streams. Increase in stochastic bit streams will eventually increase the computation time required to process these bits. These networks can be accelerated in GPU to achieve better accuracies with a significantly reduced computation time compared to CPU as the operation on these bits are bitwise and can be performed in parallel steps.

Chapter 2

Design Methodology

2.1 Overview

Implementation of a Stochastic ELM involves several steps like generation of stochastic bits, basic arithmetic computations, activation function and conversion of stochastic bits to decimal number. Following sections will explain in detail about the theory behind these blocks and need for these operations. As this work mainly focuses on creating a platform to accelerate the testing and implementation of this process, the design methodology for a conventional CPU implementation and a parallel implementation in GPU will be discussed in detail by addressing the advantages and disadvantages of each of these implementations. Throughout the work bipolar stochastic format as mentioned in section 1.3.2 is used for representing the stochastic bit streams in order to represent both negative and positive numbers.

2.2 Basic Blocks of Stochastic ELM

2.2.1 Stochastic Number Generator

Stochastic Number Generator (SNG) is one of the fundamental components of stochastic computing [9]. SNG converts a binary number to an equivalent stochastic bit stream. There are few different ways to generate SNG. One of the basic methods is by comparing the binary number with random numbers to generate stochastic bits.

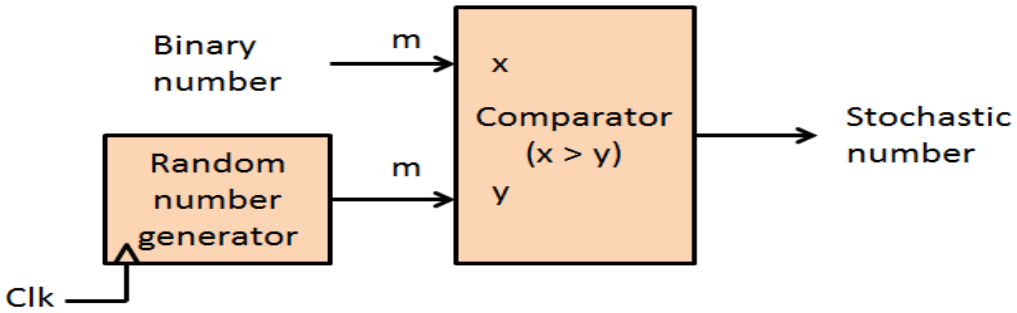

Figure 2.1: Block diagram for Stochastic Number Generator

This method is simple and less time consuming despite of having correlation errors [9]. In the following sections we will look into the implementation of SNG in CPU and parallel implementation of the SNG.

CPU Implementation

Basic SNG consists of a comparator block with two input paths. Binary number of bit length 'm' are fed into one input path and a random number of same bit length 'n' is fed into another input path of the comparator at every clock pulse. Random number generator is constructed to generate random number at every clock pulse. two types of random generator can be used, uniformly distributed and normally distributed random number generator. Uniformly distributed numbers have equal probability of happening in an interval, in this case between the range [0,1], where as the normally distributed numbers have the probability of happening more closer to the mean following a bell curve. Thus the accuracy obtained from using a uniformly distributed random numbers is better than using the normally distributed random numbers for stochastic bit stream generation. Comparator block takes the two inputs at every clock cycle and checks if the binary number is greater than the random number as shown in Figure 2.1. '1' is given as the output of the comparator if the condition is true else a '0'. This process is repeated 'n' number of times, where 'n'

is the length of the stochastic bit stream that need to be generated. For example: let the binary number be 'x', the stochastic number length needed be 'n'. let the random number be represented by y_1 for first clock pulse, y_2 for second and so on till y_n for n^{th} clock pulse. At every clock pulse, 'x' is compared with the corresponding random number to give the corresponding bit. To obtain an n-bit stochastic number, 'n' times the same operation has to be performed. This will become a tedious and time consuming process when performed in sequence for large stochastic bit streams and around 2^{12} bit length is required to obtain an acceptable accuracy. SNG is a bit wise operation and there are no inter-dependencies between the generation of the stochastic bit stream, so it can be computed in parallel using a GPU.

GPU Implementation

GPU implementation of the SNG consists of two input paths, the decimal number for which the equivalent stochastic number has to be calculated and the random number. In GPU all the random numbers are compared at once with same decimal number to generate a stochastic bit stream equivalent to the decimal number. When the condition is true output is '1' else '0'. Random numbers are distributed uniformly between the interval [0,1]. For example: when decimal number 0.5 is compared with the equally distributed 10 random number will be greater than approximately 50% of the 10 random numbers. Thus giving an output of approximately five 1s and rest of them as zero.

2.2.2 Activation Function

There are many types of activation functions like sigmoid, tanh, relu, etc used in neural networks for mapping inputs to output space. Sigmoid activation function will be used it this thesis as shown in equation 2.1. This process involves division, addition and exponential functions, which will occupy significant area and power.

These complex operations can be replaced by simple thresholding function, which will give results similar to a sigmoidal activation function [16].

$$f(x) = \frac{1}{1 + e^{-x}} \tag{2.1}$$

CPU Implementation

Sigmoid-like activation function [16] is used in this thesis for the activation of the stochastic bit streams. Let the input stochastic bit streams entering into the activation function are p_1, $p_2 ... p_m$ which are summed bit wise and compared with the threshold value equal to half of the total number of inputs. If the sum is greater than the threshold value, '1' is given as output otherwise a zero as shown in equation 2.3 and 2.4. This involves 'n' operations to count the 'n' bit parallel input streams entering the activation function. Long bit streams are required to get good accuracy. Computation time will increase with increased number of bits. Solution to increase the speed is by parallelizing the whole operation where the count of number of '1' in the corresponding positions of the bit streams does not have any inter dependencies.

$$Y(t) = g(s(t)) = g(\sum_{i=0}^{m-1} X_i(t)) \tag{2.2}$$

$$g(s(t)) = \begin{cases} 1, & \text{if } s(t) \geq \frac{m}{2} \\ 0, & \text{otherwise} \end{cases} \tag{2.3}$$

GPU Implementation

GPU implementation of activation function involves two steps of parallel operations. First operation is the count of '1's in the corresponding positions for all the input bit streams. This involves creation of 'n' threads to perform a count on n-bit streams.

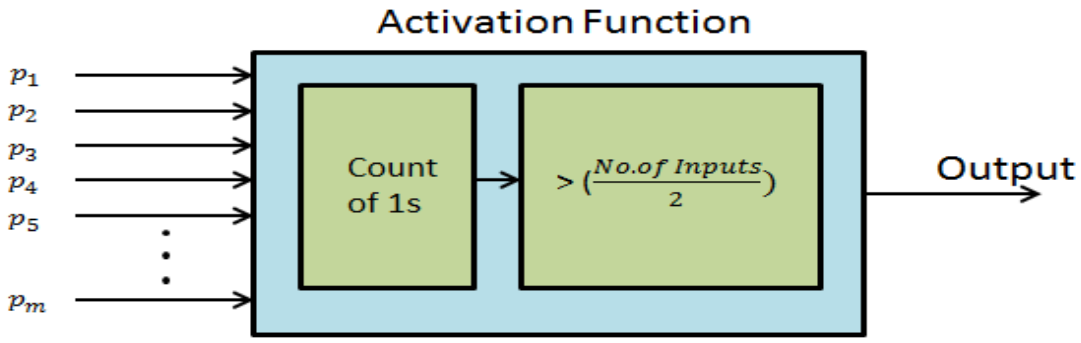

Figure 2.2: Block Diagram for Stochastic Sigmoid-like Activation Function

Output of this block consists of count values ranging from '0' to total number of inputs. This is fed into the thresholding block as shown in Figure 2.2. In one parallel step with 'n' threads corresponding to 'n' bit stream will perform a comparison with half of the total number of inputs to give '1' if this condition is true else a '0'. This step is performed to push the value to '1', when more than 50% of the input bits are active. Output of this block consists of n-bit stochastic bit stream equal to the sigmoidal output of the input bit streams

2.2.3 Arithmetic Block

2.2.3.1 Multiplier Block

Multiplication is one of the essential steps in neural network and also a complex operation when it comes to implementing traditional multipliers in hardware. For example: N bit combinational multiplier requires N^2 AND gates, N-1 half adders and 2N-1 full adders [27]. Thus these multipliers occupy large space and consume more power due to its complex circuitry. Where as a stochastic multiplier uses a 2-input XNOR gate for the multiplication of two stochastic bits in bipolar format. Makes the whole process of multiplication more area and power efficient.

CPU Implementation

Multiplication operation can be performed by using AND operation on two single line unipolar stochastic logic and XNOR operation can be used to perform multiplication of two single line bilopar stochastic logic [28]. The multiplier circuit will consist of XNOR gate with two inputs to perform a bit wise operation on two stochastic bit streams where the XNOR gate will push the stochastic bit stream output to negative when the input bit streams contain a combination of almost equal number of '1's and '0's else it will push it to a positive extreme when most of the bits are '1'. Thus a signed multiplication is supported by an XNOR gate. one of the major disadvantage is the drop in accuracy compared to a conventional method. This is due to the lack of precision, which can be increased by increasing the length of the bit stream [9] and will lead to long computation time of around 'n' clock steps for n-bit operation. Stochastic multiplication is again a bit wise operation with no inter dependencies. This feature can be utilized to parallelize the multiplication operation using a GPU.

GPU Implementation

Implementation of multiplication in GPU involves 'n' number of threads performing XNOR operation on '2n' bit streams that need to be multiplied based on the corresponding positions of the bits in the stream, provided both the bits are of same bit length. This implementation will reduce the 'n' step operation time to one parallel operation and gives more speedup when many multiplication operations need to be performed. This can be achieved by arranging the bits in the corresponding position that need to be multiplied.

2.2.3.2 Subtraction Block

Binary subtraction is one of the essential arithmetic operation in any digital circuitry. Digital implementation of ANN requires a subtractor for finding the difference be-

tween the predicted and target output. Implementation of binary subtractor requires 'n' full adders. Thus occupying large chip area and power. One of the simplest ways to implement this will be using a stochastic subtractor consisting of a 2: 1 MUX and an inverter circuit. This will reduce the chip area, cost and power consumption compared to a conventional subtractor shown in Figure 2.3.

CPU Implementation

To subtract two Stochastic bit streams x and y, one of the operands should be inverted bit wise and then fed into a 2:1 MUX along with the other operand to get the difference. To obtain x - y as shown in the Figure 2.3, Bit stream y is fed into an inverter for a bit wise inversion to obtain -y. x and -y is fed as input to the 2:1 MUX and the select line is a stochastic bitstream equal to 0.5 in a unipolar stochastic format. There is a drawback in using a MUX to subtract two numbers as it scales down the value by two due to the select line value equal to 0.5. Half of the sum of the two inputs are only obtained as output. This method holds good for addition or subtraction in stochastic representation as the output should be between '0' and '1'. Eventually there is some accuracy loss in these blocks. Another drawback is the computation time involved in the bitwise inversion of bits followed by a MUX operation. A total of 2n steps are required for an n-bit operation. Obvious solution to this is to parallelize both the operations and perform the whole operation in two parallel step.

GPU Implementation

Subtraction operation of stochastic numbers can be made in parallel by using the boolean equation of MUX operation, which is given in equation 2.4, where Z is the output, S is the select line, x and y are the inputs. In order to subtract two stochastic numbers, one of the inputs needs to be inverted. Final subtraction can be achieved by

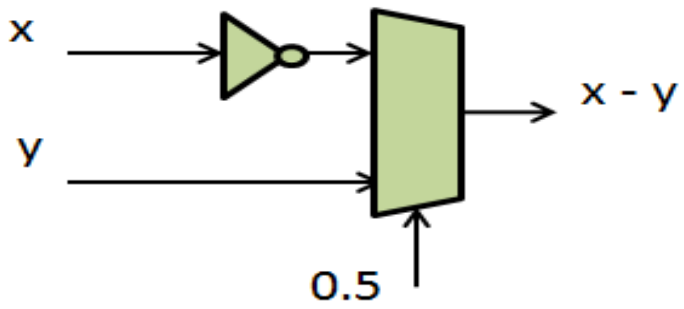

Figure 2.3: Circuit diagram for CPU stochastic subtraction

modifying equation 2.4 to give equation 2.5. The whole process can be performed in three parallel steps which are inversion, AND operation followed by an OR operation.

$$Z = (x.S + y.S') \tag{2.4}$$

$$Z = (x.S + y'.S') \tag{2.5}$$

2.2.4 Stochastic to Decimal Number Conversion

Stochastic to decimal number conversion is one of the basic elements in SC [9]. This is performed by counting the number of 1s in the bit steam and dividing it by SNL (total bit length) for unipolar stochastic format and for a Bipolar Stochastic format the count of 1's divided by SNL is added with 1 and divided by 2. These conversion blocks have large overhead and are the most expensive blocks in SC. This disadvantage can be overtaken by the benefits achieved in the implementation of other arithmetic blocks.

CPU Implementation

The stochastic to decimal converter can convert only one stochastic bit stream at a time. It consists of a counter for counting the number of 1's followed by a division

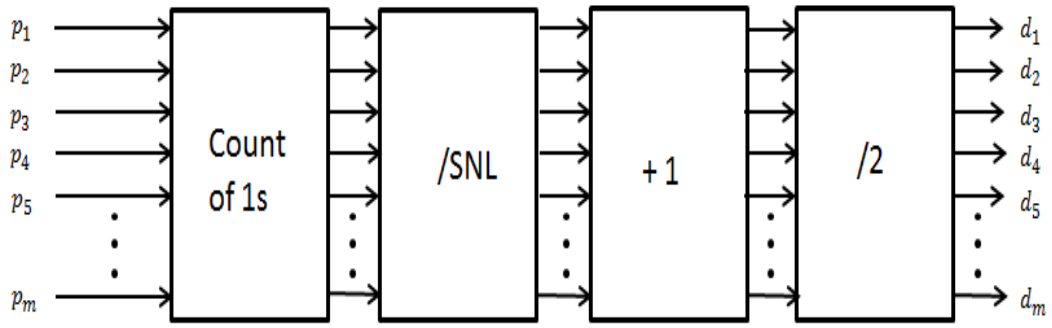

Figure 2.4: GPU implementation of stochastic to decimal conversion

block for dividing the counter output by SNL, followed by an adder to add 1 and then divide by 2. These 4 steps of of sequential operation are parallelized to increase the speed of operation. Let the number of stochastic numbers to be converted be m, then the total number of steps will be (n X m X 4). The speed of operation can be increased by performing parallel operation on all the stochastic numbers at a time.

GPU Implementation

GPU implementation of this block can be performed in 4 parallel steps as shown in Figure 2.4. p_1 to p_m are the stochastic numbers that need to be converted. In one parallel operation, The counter counts all the 1s in all the stochastic bit streams, then it is divided by SNL, added to 1 and again divided by 2. Thus the number of operations reduce from (n X m X 4) to just 4 parallel steps, thus decreasing the computation time of these bits.

2.3 Summary

The stochastic implementation of NN in CPU and GPU are compared and studied to arrive at an optimal architecture in terms of computation time. GPU implementation has proven to significantly reduce the computation time in processing stochastic bit streams as the operations on the same are bitwise.

Chapter 3

Accelerating Stochastic ELM in GPU

3.1 Overview

Parallel implementation of stochastic ELM is implemented in GPU. In this work the GPU is called from MATLAB using the parallel computing tool box with 6.1 compute capability. GeForce GTX 1050 Ti is a Nvidia graphics card used in this implementation which has 768 cores and 4 GB memory configuration. GPU implementation in MATLAB is chosen for its simplicity. The operation can be performed by porting the data from the MATLAB workspace to GPU memory by storing it in GPU array. Any operation performed on this data will be parallelized and will behave like an operator overloaded function. The following sections will explain in detail about the layer by layer implementation of stochastic ELM in GPU.

3.2 Input Layer

The input layer consists of the input nodes equivalent to the number of features. This layer can be accelerated by using two GPU blocks which generate the input and weight stochastic bit streams, (i) decimal to bipolar stochastic number converter (DBC) and (ii) stochastic number generator (SNG). DBC block converts all the decimal numbers from [-1,1] to [0,1] range in 2 parallel operations by adding one to the decimal number in first step and then dividing it by 2 in the second parallel step. Relatively, a CPU

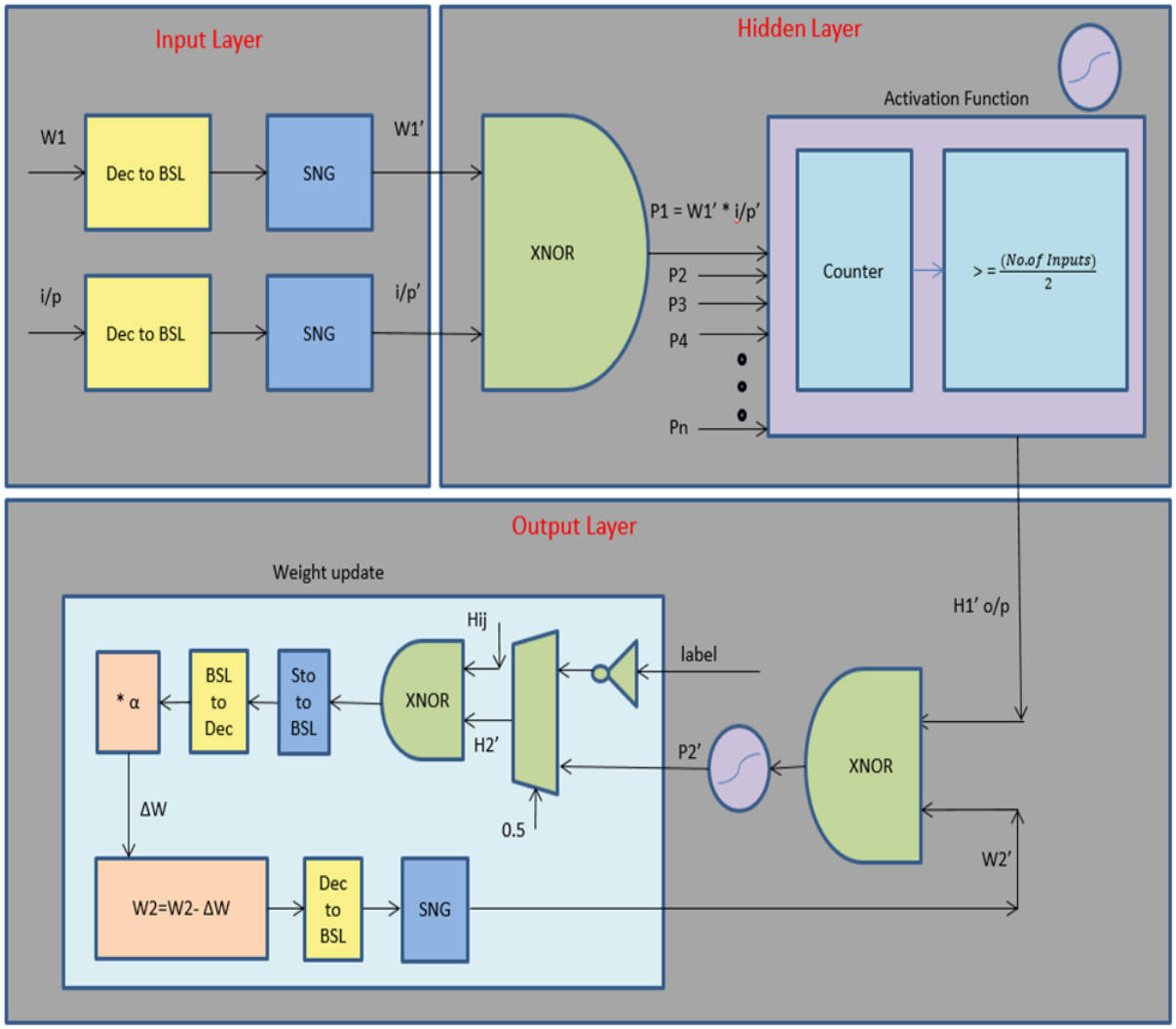

Figure 3.1: Architecture of stochastic ELM

will take 2 X M steps where a 2D array is maintained to store the BSL values.

3.2.1 Stochastic Number Generation

SNG designed in this work is based on one of the standard methods, which compares the binary number with the random numbers to generate stochastic bits. To generate M stochastic numbers with N bits, it requires M X N steps [9]. In this work, the above mentioned method is modified to work in one parallel step in GPU to obtain all the input nodes simultaneously by using a two level parallel SNG structure, the levels are (i) generation of stochastic bits for all the input nodes in parallel and (ii)

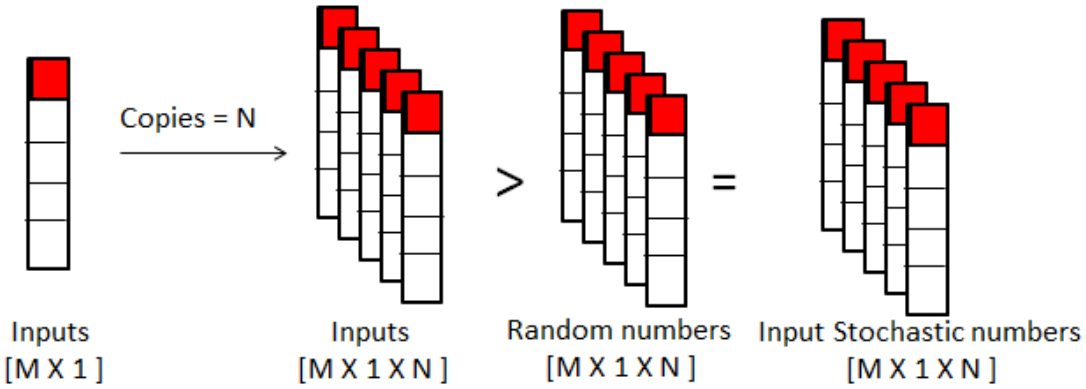

Figure 3.2: Visual representation of GPU implementation for the generation of input stochastic bit stream, 'M'= number of input layer nodes and 'N'= stochastic bit length

generation of N stochastic bits for each of these input nodes in parallel as shown in Figure 3.2. Two inputs to the SNG are the BSL matrix and random number matrix. BSL matrix is converted to a 3D matrix of size [M X 1 X N] by making N copies of the 2D BSL matrix and a random number matrix of same size with range [0,1] has been generated. These two matrices are positionally mapped and given as an input to the SNG to generate an output matrix of same size. SNG consists of a comparator, which gives an '1' when the value in BSL matrix is greater than the random number matrix. The red boxes in the Figure 3.2 indicate the generation of stochastic bit stream for the first input node.

Totally three steps generate all the input nodes in the GPU, whereas it requires (2 X M) + (M X N) = M(2 + N) steps in the CPU. Next step in the network is to generate the synapses (weight values). In an ELM, similar to the input node generation, stochastic bit stream for weights can be generated in three parallel steps (2 for DBS and 1 for SNG) that is independent of the network size as shown in figure 3.2. In a CPU, this implementation will require (2 X M X H) + (M X H X N) = (M X H)(2 + N) steps. Total number of computational steps for the input layer are '6' in the GPU and M(2 + N)(1 + H) in CPU.

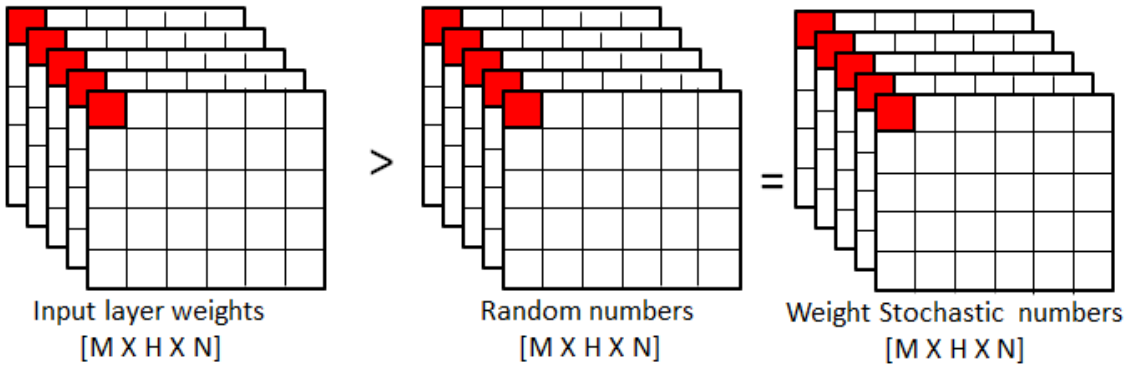

Figure 3.3: Visual representation of GPU implementation for the generation weights stochastic bit stream, 'M'= number of input layer nodes, 'H'= number of hidden layer nodes and 'N'= stochastic bit length

3.3 Hidden Layer

Generation of Hidden layer nodes require two components, they are (i) multiplier to find the product of the input and weight values for all the connections (synapse) and (ii) activation function. Stochastic multiplier designed in this work has a three level parallel structure. They are (i) bitwise XNOR operation of input and weight stochastic bit streams (ii) stochastic multiplication of one input with all the weight values and (iii) multiplication of all the inputs and weights. Whole operation takes (M X H X N) XNOR bitwise operations with no inter dependencies. The input matrix from the input layer is converted from [M X 1 X N] to [M X H X N] size by making H copies horizontally. This input matrix is positionally mapped with the Weight matrix of same size to carry out the parallel XNOR operation on both the matrix to obtain the product matrix in one parallel step as shown in Figure 3.4.

3.3.1 Activation function

The stochastic activation function designed in this work is based on the sigmoid-like function. The input stochastic bit streams entering the hidden node are summed bit wise and compared with the threshold value equal to half of the total number of

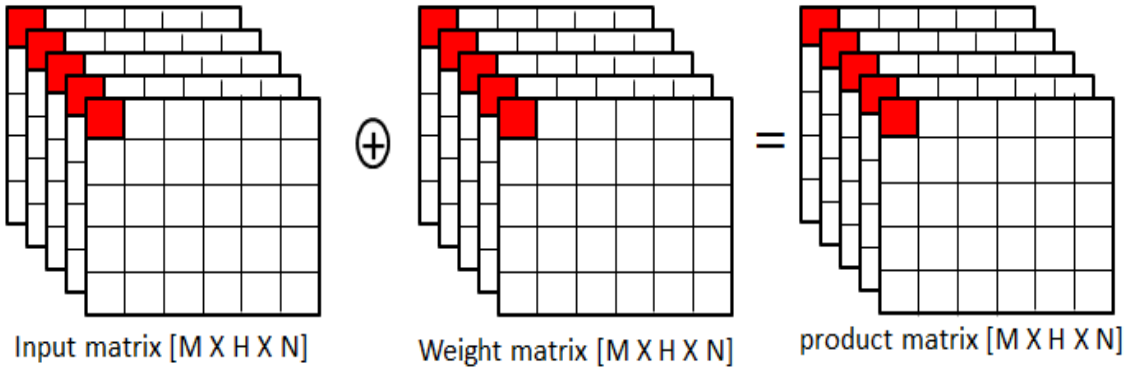

Figure 3.4: Visual representation of GPU stochastic multiplication using XNOR gate, 'M'= number of input layer nodes, 'H'= number of hidden layer nodes and 'N'= stochastic bit length

inputs. If the sum is greater than the threshold value, 1 is given as output otherwise a zero[16]. Total number of steps involved are 2N, which includes N summation and N comparison operations for N bits. The proposed design consists of a parallel structure which performs the summation and comparison in one parallel step as shown in Figure 3.5. 3D product matrix of size [M X H X N] is given to the GPU to generate a summation matrix of size [1 X H X N]. This matrix consists of values ranging from '0' to the number of hidden layers. This matrix is compared with the N/2 in one parallel step to give the hidden layer output matrix. The activation function takes 2 steps to generate the hidden layer matrix from the product matrix. The total number of steps to generate the hidden nodes is four, where as in the CPU it is (M X H X N) + 2(H X N).

3.4 Output Layer

Output layer is initialized with random weights and then updated for each epoch. Stochastic number generator is used to generate these weights in one parallel step and followed by the GPU multiplier to get the stochastic bit streams for the output layer

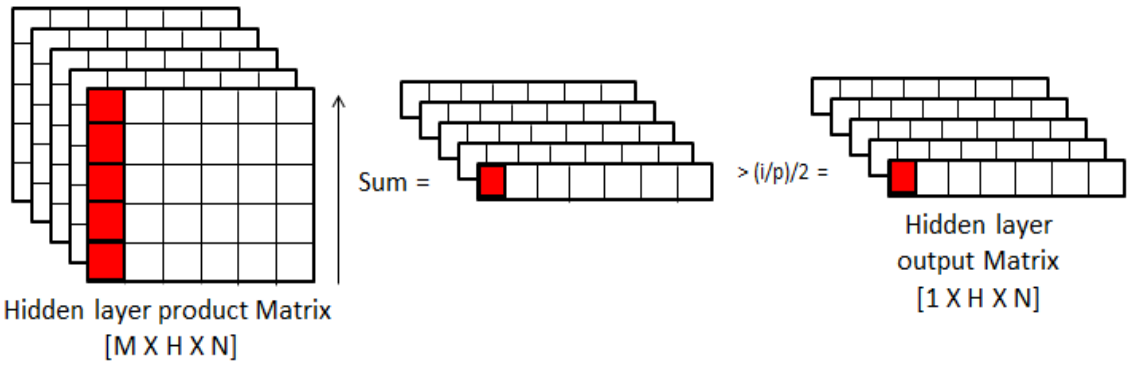

Figure 3.5: Visual representation of GPU implementation for Stochastic Sigmoid-like Activation Function, 'M'= number of input layer nodes, 'H'= number of hidden layer nodes and 'N'= stochastic bit length

3.4.1 Stochastic Training Circuit

A novel GPU stochastic training circuit has been designed to perform the weight update operations as shown in Figure 7. This circuit is based on the delta update equation derived from the least mean squares [29]. ΔW is the value by which the weight needs to be updated.

$$\Delta W = (y'^{(k)} - y(k))h(m) \tag{3.1}$$

The subtraction of the hidden layer output from the input labels uses a 2:1 MUX. Subtraction output is achieved in parallel by using the MUX operation, as shown in equation 5; where Z is the output, S is the select line, x and y are the inputs. In order to subtract two Stochastic numbers, one of the inputs need to be inverted. The whole process can be performed in three parallel steps: inversion, AND operation followed by an OR operation.

$$Z = (x.S + y.S') \tag{3.2}$$

$$Z = (x.S + y'.S') \tag{3.3}$$

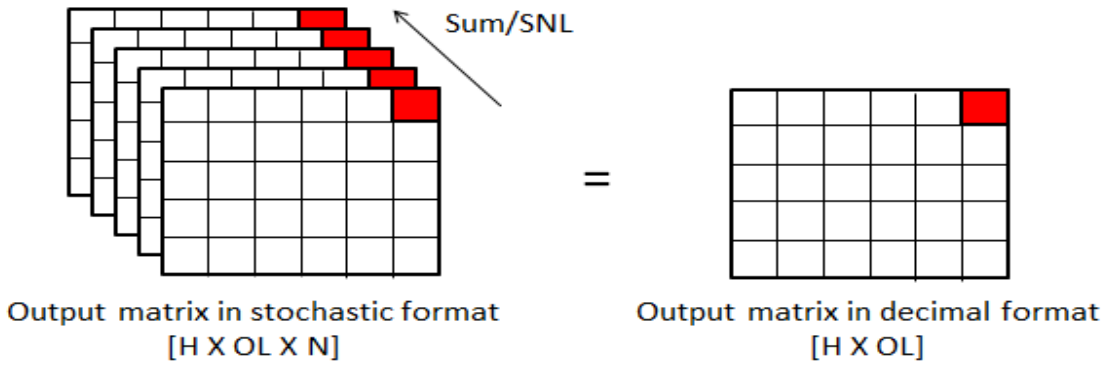

Figure 3.6: Visual representation of GPU implementation for stochastic to decimal conversion, 'OL'= number of output layer nodes, 'H'= number of hidden layer nodes and 'N'= stochastic bit length

In a CPU, 2(N X H) operations will be required for inversion and MUX operation of H hidden nodes with N stochastic bits.

The difference between the target and predicted value has to be multiplied by learning rate of the network alpha. It is the rate at which the network learns to update the synaptic weights through the iterations. To convert this alpha value to stochastic bit stream with good precision requires more number of bits throughout the system. To avoid this, a stochastic bit stream is converted to decimal number before multiplying with the alpha value. In Figure 7, H2' represents the output of a multiplexer fed as input to the GPU stochastic to BSL conversion block (SBC). This block counts the number of 1's in each bit stream and divides the sum by SNL in two parallel operations. BSL then converts to decimal number of range [-1,1] using the BSL to decimal converter. All the elements of the output matrix from this block are multiplied by alpha value. Output of this block gives ΔW. The weights are updated by subtracting ΔW from W2. Output of this block is converted back to BSL format and sent through SNG to get the stochastic bit streams for weights between the hidden and output layer. The total number of steps taken for training are eleven in a GPU. In a CPU the total number of steps will be equal to (N X OL) + (N X OL) + (2 X OL) + (2 X OL) + OL + W2 + (2 X W2) + (W2 X N)=(2 X OL)(N + 2

OL + 1/2) + W2(1 + 2 + N), where, the number of output nodes are represented by OL and the number of hidden layer weights by W2 as shown in Figure 3.6.

3.5 Summary

The implementation of stochastic ELM in GPU was implemented, an accuracy of 92.4% for MNIST dataset and 87.5% for orthopaedic dataset. It has been observed that the maximum parallelization can be performed layer by layer only, as the input and output of the layers are interdependent.

Chapter 4

Accelerating Stochastic ESN in GPU

4.1 Overview

The stochastic implementation of ESN has been accelerated in GPU. The input and output layer has been implemented similar to the input and output layer of stochastic ELM explained in chapter 3.

The reservoir layer in ESN has been implemented differently compared to the hidden layer in stochastic ESN . Also a time series dataset has been used to train and test the network whereas a Multivariate dataset has been used in Stochastic ELM. Following sections will address these differences in the implementation of this network from Stochastic ELM as shown in Figure 4.1 for easy understanding.

4.1.1 Reservoir layer

The reservoir layer is the essence of the ESN network which serves as a memory to store the values from previous state, thus able to classify time series data. First step is to calculate the reservoir layer node values, which gets updated at each incoming input. For the first incoming input of each sample, the reservoir layer values are calculated by the stochastic multiplication of input and weight matrix as discussed in chapter 3.3. From second input, the previous value of the each hidden node is used and updated for every new incoming input. Reservoir node values are multiplied with

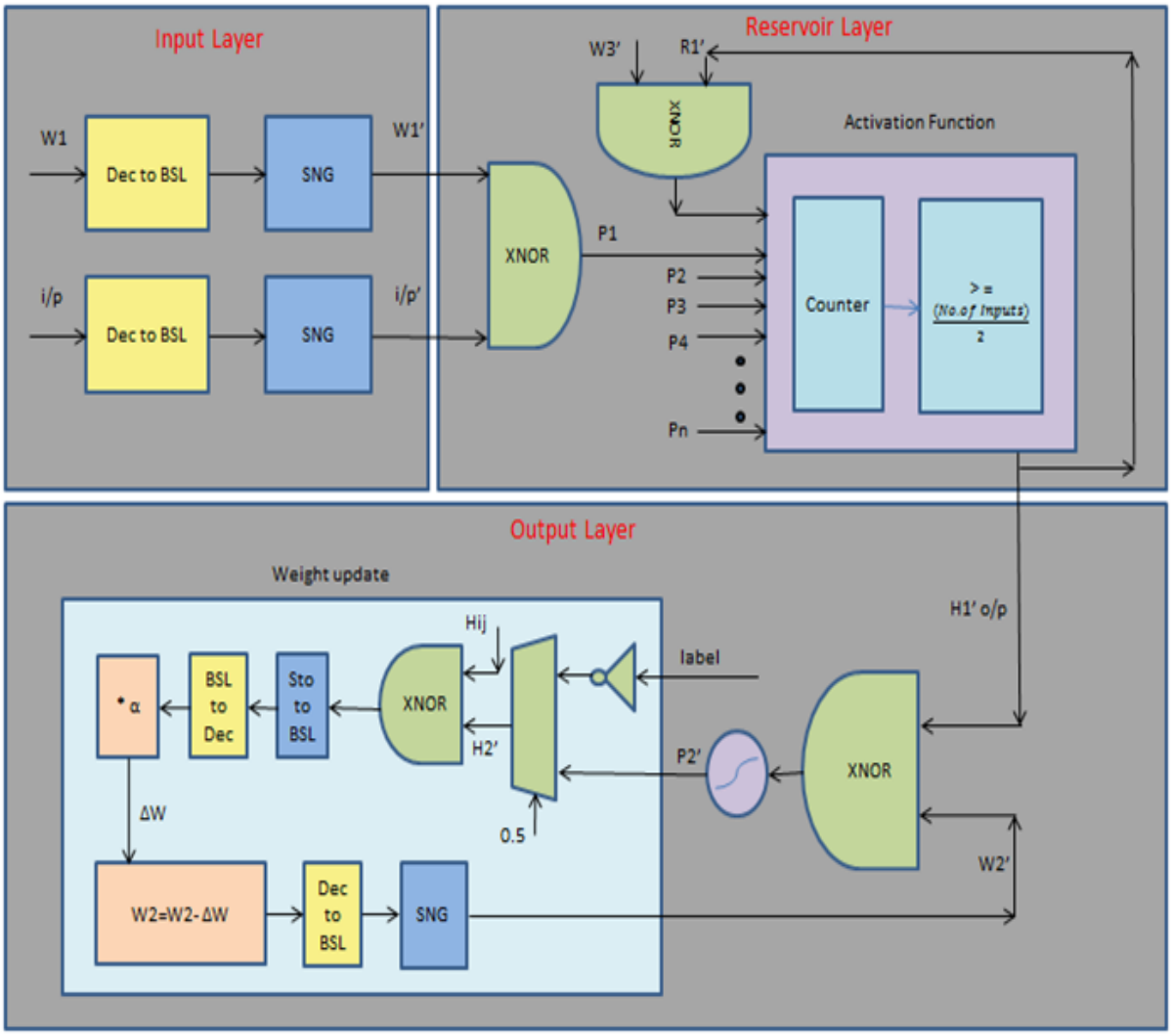

Figure 4.1: Architecture of a stochastic ESN

the weight values to generate the "syn_connect" matrix containing all the synaptic connections between the reservoir nodes of size [H X H X SNL], where H is the number of reservoir nodes. Once we have all the connection values, we can activate the connections between the nodes either in random order or in a particular format such as random connection based on the percentage of connection, ring, center node, hybrid, 2D mesh, etc [30]. In this work the random topology and ring topology for reservoir layer connection has been implemented.

4.1.1.1 Random topology

Random connection percentage is taken as input (R%) and a matrix containing 1's equal to R% of total number of reservoir layer nodes (H) is generated equal to [H X H] as shown in Figure 4.2. In this example, there are 6 reservoir layer nodes (H1, H2 to H6) and the input percentage of connection is 50%. Exactly 3 out of 6 values in each row is a 1 (3/6 = 0.5). This matrix is randomly generated using the stochastic number generator explained in chapter 3.2.1.

	H1	H2	H3	H4	H5	H6
H1	0	1	1	0	0	1
H2	1	0	1	0	1	0
H3	0	1	0	1	0	1
H4	1	1	0	0	1	0
H5	0	1	0	1	0	1
H6	1	0	1	1	0	0

Figure 4.2: Matrix generated to activate the connections between the reservoir layer nodes for 50% connection

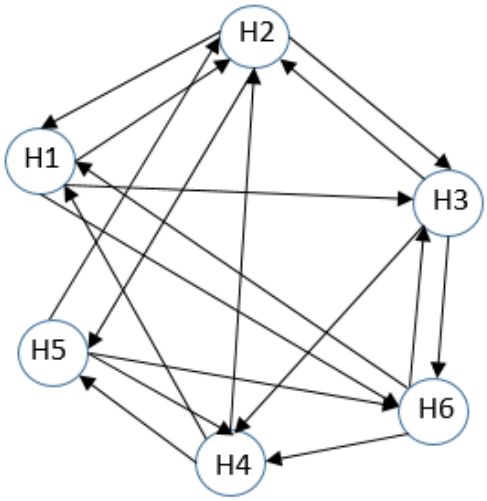

Figure 4.3: Connections between the reservoir layer nodes based on the randomly generated matrix

Each row consist of R% of 1's and rest equal to '0'. This matrix is used to randomly activate the connections in the reservoir. '1' denotes the presence of a connection and '0' represents no connection from reservoir layer node in X axis to the corresponding reservoir layer node in Y axis. All the diagonal values are converted to zero to prevent the connection to the same node from the previous stage. Based on this matrix the connection between the reservoir layer node is shown in Figure 4.3. This is a representation in 2D, in order to implement the same operation for the stochastic bits, the Connection activation matrix has to be copied SNL number

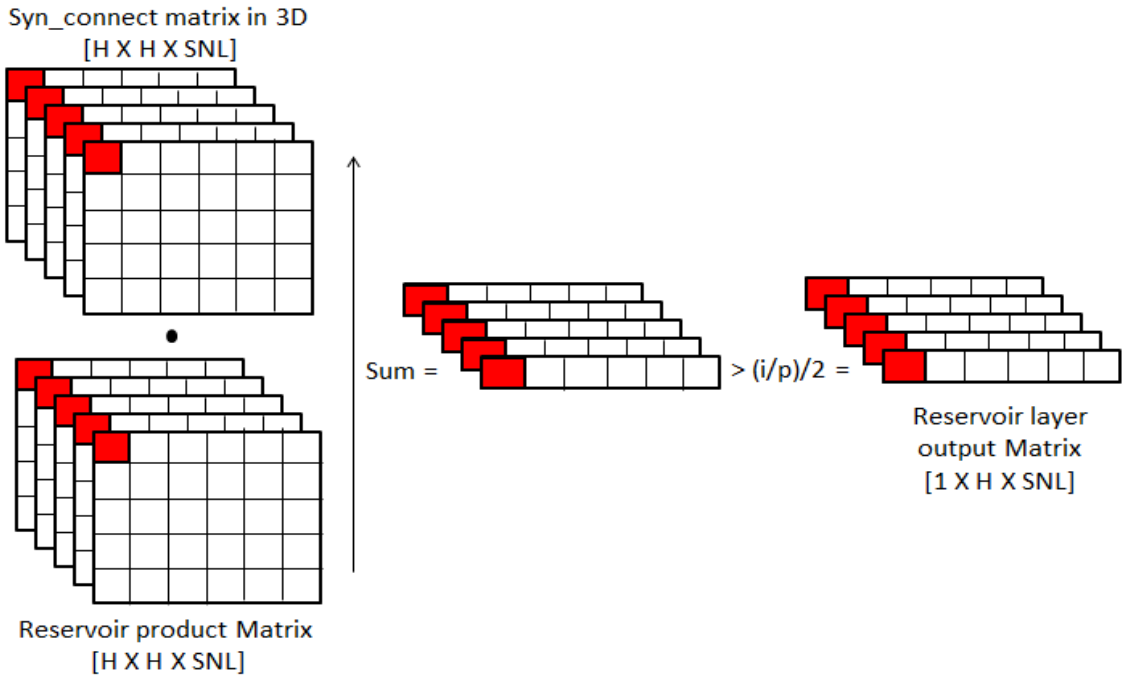

Figure 4.4: Stochastic GPU implementation of reservoir layer with random topology, 'H'= number of hidden layer nodes and 'SNL'= stochastic bit length

of times. The matrix is converted from [H X H] to [H X H X SNL]. The current input values calculated are added to the matrix and also the corresponding rows and columns in connection activation matrix are set to value '1'. so that the current input values are always actively connected during calculation. AND operation is performed on the 3D connection activation matrix and the syn_connect matrix as shown in Figure 4.4 (discussed in chapter 4.1.1). This way only the activated connections will have the original values and the rest of the values are converted to zero. Threshold value for the activation function is half of R% of total no. of reservoir layer nodes. This way the output of each reservoir node is calculated.

This method is very uncertain and the best result can be narrowed down only by trial and error method. Due to the random nature, stochastic implementation of this method doesn't produce a good accuracy.

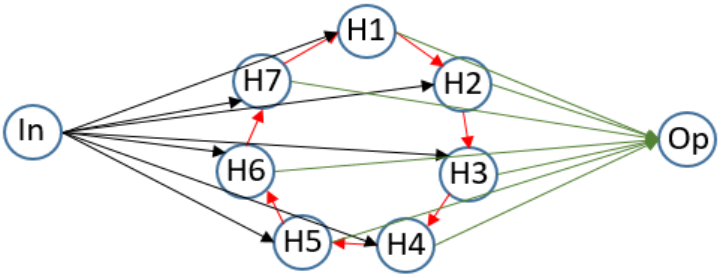

Figure 4.5: Example Ring topology connection

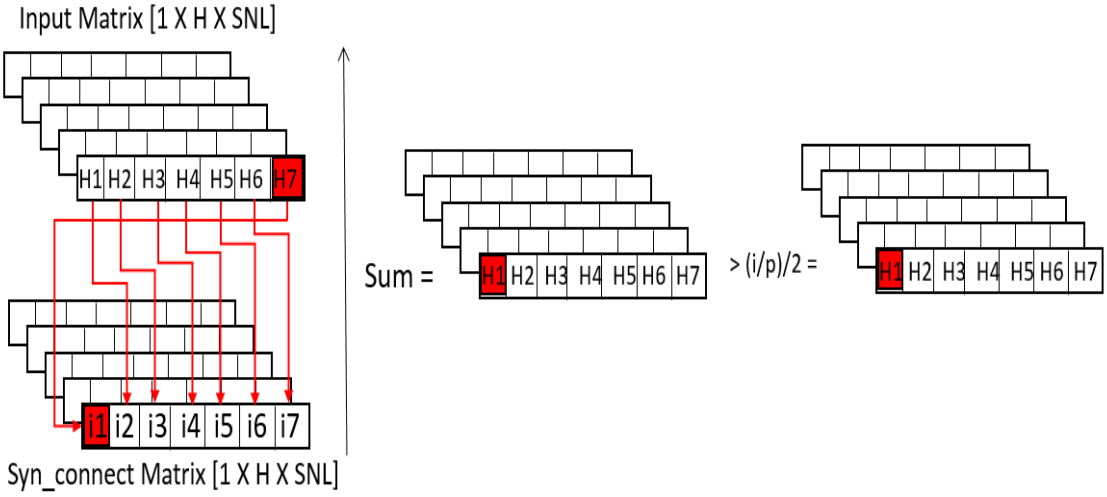

Figure 4.6: Stochastic GPU implementation of reservoir layer with ring topology

4.1.1.2 Ring topology

In ring topology, the reservoir layer nodes are connected in a circular format as shown in Figure 4.5. This is much simpler to implement and also gives a better accuracy due to the lack of randomness in the connection between the nodes in reservoir layer compared to the previous method. Here the output of H1 multiplied with the weight value of the connection is given as an input to H2, similarly from H2 to H3, H3 to H4 and so on. All the input and output nodes are connected to all the reservoir layer nodes.

To calculate the current value of the reservoir nodes the 'syn_connect' matrix from previous state and the current input matrix is mapped as shown in Figure 4.5 and

sent through the activation function (summation of the two matrix is mapped and then compared to half of the total no. of inputs, which is 2 in this example). No. of steps taken in GPU at the reservoir layer are one step for multiplication of weight values and reservoir node values, one step for mapping, one step for summation and one step for thresholding. Total GPU time taken = 4. whereas in CPU, it takes [H X H X SNL] for multiplying the weights values and the reservoir layer values and 2 times [i/p X H X SNL] times for summation and thresholding of each bit. Total CPU time = [H X H X SNL] + 2[i/p X H X SNL] = H X SNL [H + 2(i/p)].

4.2 Summary

GPU implementaion of stochastic ESN to detect the epileptic seizure from a EEG signals achieved an accuracy of 73.5%. Ring and random topology are studied and implemented in GPU to accelerate the reservoir layer. The ring topology is less complex and faster in GPU compared to random topology.

Chapter 5

Result Methodology

The main goal of this work is to implement a stochastic logic ELM and stochastic logic ESN using single line bipolar representation for useful application that can take advantage of the immanent properties of the network like energy efficiency and better computational capabilities.

5.1 Application

In this work, written number recognition and orthopaedic dataset (to detect abnormalities in orthopedic) are tested on Stochastic ELM and epileptic seizure detection is tested on stochastic ESN.

5.1.1 Orthopaedic dataset

Orthopaedic data set is of type Multivariate, which was built by Dr. Henrique Da Mota and donated in the year 2011 [31]. This data set classifies orthopedic patients into normal or abnormal category and the patients falling under abnormal category has either disk hernia or spondilolysthesis. Disk hernia also called as slipped disc, where the soft and central portion bulges beyond the fibrous ring of an inter-vertebral disc affecting the spine [32] as shown in Figure 5.1. The dataset consists of six biomechanical attributes of each patient , that can be used to detect the disease, based on the orientation and shape of lumber spine and pelvis, they are - pelvic tilt, pelvic

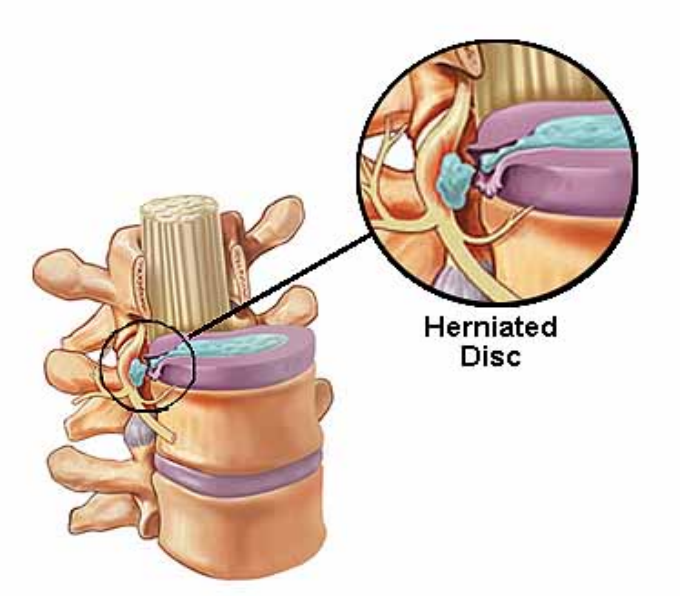

Figure 5.1: Sample images of MNIST dataset [1]

radius, grade of spondylolisthesis, pelvic incidence, sacral slope and lumbar lordosis angle [31].

A total of 400 train and 100 test samples are used in this network containing a mixture of normal and abnormal patients. The number of weights required is minimal as there are only two output classes. These input samples are normalized to the range [-1,1] and later converted to single line bipolar format to implement the network. These kind of applications will be of great use in the medical field to classify patients based on other abnormalities.

5.1.2 Written Number Recognition

The MNIST dataset which stands for modified national institute of standards and technology, consist of images of numbers from '0' to '9' as shown in Figure 5.2. These are handwritten by a variety of people like high school students, census bureau employees, etc. [33] MNIST dataset consist of 60,000 training and 10,000 testing samples, each containing 784 features (28 X 28 grey images). Feature extraction has been performed on this data in MATLAB to down sample it to 25 features for each

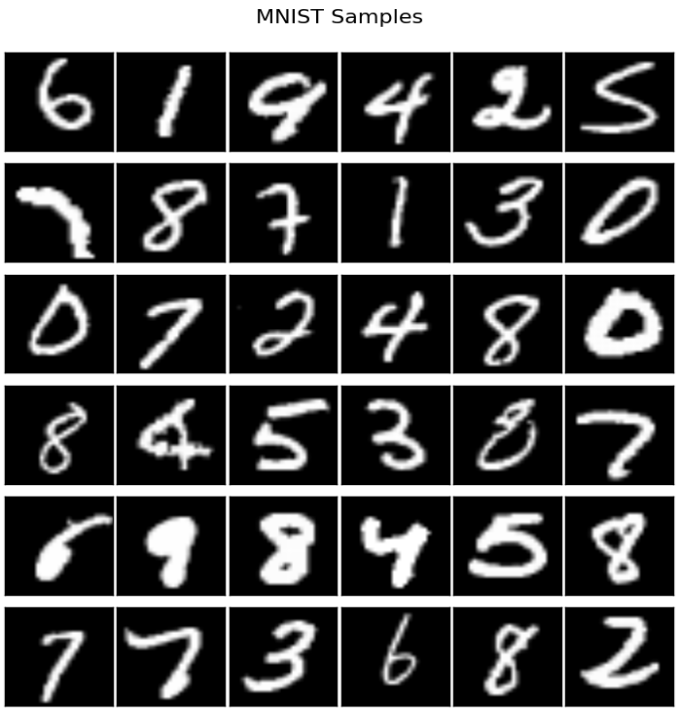

Figure 5.2: Sample image of a patient containing disk hernia [2]

input (for each image of the number). Total of 800 train data and 200 test data has been picked as the input for the stochastic ELM network.

Since the network uses stochastic logic, all the inputs have to normalized to probability values ranging from [-1,1], which is later converted to single line bipolar format of range [0,1]. There are 10 classes of data that has to be classified thus increasing the number of weights required. Also increasing number of hidden nodes required to get a good accuracy on classifying this data.

5.1.3 Epileptic Seizure Detection

Epilepsy is neurological disorder which affects 68.8 in every 100,000 people [34]. One third of the people affected by epilepsy have seizures even after getting treated with the new medications. If we can detect seizure at an early stage, patients can be given targeted treatments for seizures [35]. This will prevent from unwanted accidents, drug

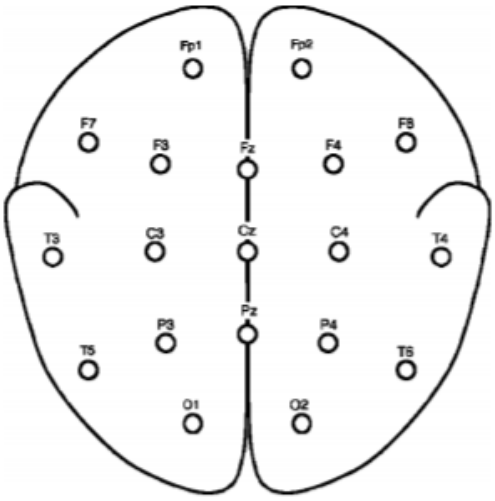

Figure 5.3: Surface electrode placement for EEG signal Set A recording [3]

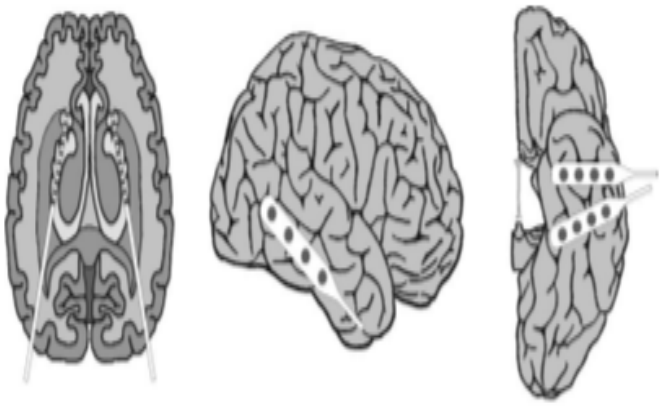

Figure 5.4: Surface electrode placement for EEG signal Set E recording [3]

over doze and will be able to provide a monitored treatment. Seizure occurs due to abnormal electrical activity in the brain, this will lead to uncontrollable shaking of the body and even unconsciousness in some cases [36]. Seizure can be detected using electroencephalogram signals (EEG), which are measured from the patients brain using the implanted electrodes.

One such dataset is the EEG dataset published in the year 2001 [3]. Here the electrical properties of different brain states and regions were compared and recorded for healthy volunteers and patients with epilepsy. Dataset consists of 5 sets of 100

Figure 5.5: Example EEG time series data of a healthy volunteer without seizure from Set A [3]

Figure 5.6: Example EEG time series data of a patient with seizure from Set E [3]

single channeled EEG signals each, spanning for 23.6 secs. These 5 sets are denoted as A, B, C , D and E, where set A consists of the EEG signals of five Healthy volunteers recorded by following the standardized electrode placement scheme as shown in Figure 5.3 and set E consists of EEG signals collected from five epilepsy patients with seizure during presurgical evaluation by placing the depth electrodes into the hippocampal formations of the brain as shown in Figure 5.4. Example EEG time series data from set A for normal case and from set E with seizure for one second is shown in Figure 5.5 and 5.6 respectively.

A set of 800 EEG signals for training the network has been picked, which contains 400 signals from set A (normal case) and 400 from set E (seizure case). Similarly 200 EEG signals for testing the network (100 from set A and 100 from set E). The input data has to be normalized to the range [-1,1] and later converted to the single line bipolar format for processing the bits in stochastic logic.

Chapter 6

Results and Analysis

6.1 Speedup for Basic Blocks

6.1.1 Speedup for Stochastic Number Generator

The speedup for individual block is obtained by comparing the time consumed by individual block for computation when implemented in CPU and GPU. The stochastic number generator as mentioned in chapter 3.2.1 generates stochastic bit streams for a given binary number. Figure 6.1 shows the speedup of SNG for increasing bit length. This shows that the computation time on CPU increases in large amount with increasing bit length where as the computation time in GPU doesn't increase in such large amount thus increasing the speedup with increasing bit length. Reasons for the under performance of CPU with increasing bits is due to the time taken in processing the bits one by one in a sequence. This will linearly increase the time consumed by the CPU. In the case of GPU implementation, the computation time increases in small steps with the increase in number of bits because the bits are processed in parallel. There is a some jitters in the graph, this is due to the sudden change in the CPU computation as it is used for running other jobs in the background.

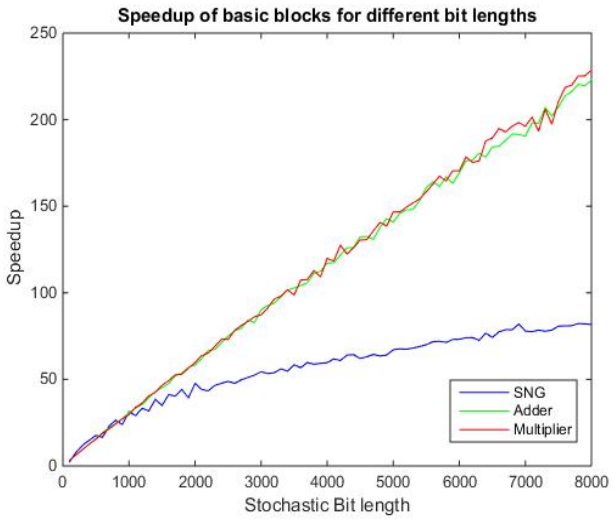

Figure 6.1: Speedup of SNG, adder and multiplier for 100 iterations

6.1.2 Speed up for arithmetic blocks

The basic arithmetic blocks of this implementation are multiplier and adder as mentioned in chapter 3.2.2 and 3.2.3 respectively. Figure 6.1 shows the speedup vs bit length plot, where the speedup is taken as the average of 100 iterations for multiplication and addition block. Over 100 iterations the blocks have proved to give an increasing speedup for the increase in bit length. For higher bit length the speedup decreases when the maximum memory limit of the GPU is reached while using Nvidia GeForce 1050 Ti GPU. As seen in the graph, the arithmetic blocks seem to give better speedup than the SNG for higher bits.

6.2 Stochastic ELM in GPU

6.2.1 Choice of Parameters

6.2.1.1 Stochastic Bit Length

Stochastic bit length is the number of bits in each bit stream, which is maintained the same throughout the network. In order to choose the appropriate bit length, mean

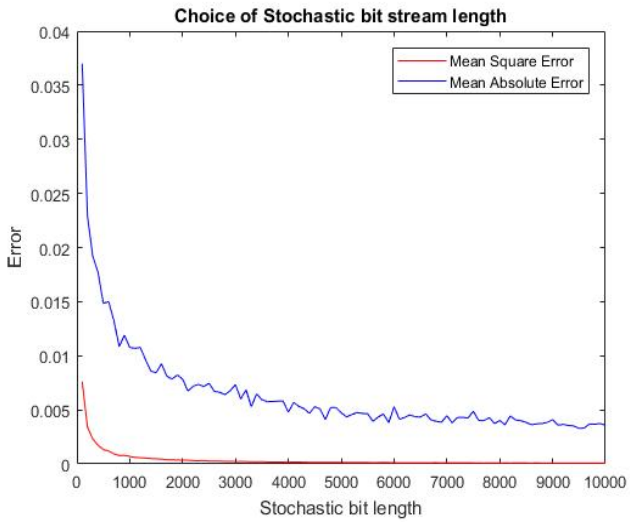

Figure 6.2: Stochastic bit length vs. MSE for MNIST dataset

square error (MSE) and mean absolute error (MAE) were plotted for the building blocks for different bit length. MSE is a plot of average of the square of the difference between the target and the predicted output. Figure 6.2 shows MSE and MAE for the stochastic number generator. The Figure 6.3 shows the MSE and MAE for the arithmetic blocks. The error caused by using MUX and XNOR is almost similar and this is due to the correlation error [9]. The plot shows that the increase in bit length mitigates this problem. With the increase in bit length, numbers can be represented with better precision. For example 0.5 can be represented in 10 bits but 0.55 requires 100 bits for representation. Thus proving the fact that more bit lengths are required for better precision. MSE reaches zero percent error after 4000 bits and MAE reaches around 0.005% error at around 4000 bit stream length as shown in the Figure 6.2. From this observation, $2^{12} = 4096$ is chosen as the bit stream length for this work.

6.2.1.2 Random number generation

Two types of random number generation were tested for the generation of stochastic bit streams - uniformly distributed random numbers and normally distributed random numbers. These random numbers are floating point numbers with 2^{16} bit precision.

Uniformly distributed numbers have equal probability of happening in an interval whereas the normally distributed numbers have the probability of happening more closer to the mean, following the bell curve [8]. One of the inputs to stochastic number generator are the random numbers, which is generated using both uniformly distributed and normally distributed random numbers and plotted for error against the increasing bit length as shown in Figure 6.3 - 6.6. This shows that the MSE calculated for normally distributed random numbers almost reaches '0' at around 2^{12} bit stream length, where as for normally distributed numbers the stochastic bit generation has an MSE percentage of 0.035.

Similarly for the plot of MAE, the normally distributed numbers have around 0.25% of error and the plot is highly undulating and unreliable as shown in Figure 6.6, where as the uniformly distributed numbers generate an MAE of 0.005% as shown in Figure 6.4. This variation in error percentage is due to the uneven distribution of numbers when generated normally. From these plots, it was concluded that uniformly distributed random numbers are more suited for the stochastic number generation and the same is used in this work.

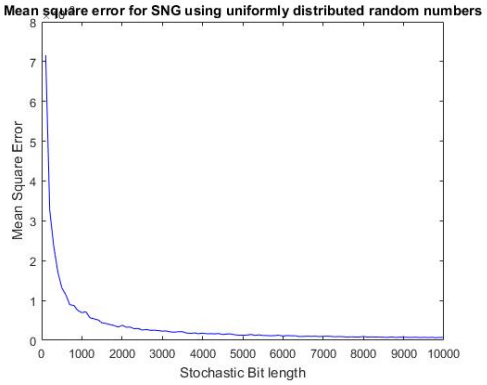

Figure 6.3: Mean square error for uniformly distributed random numbers

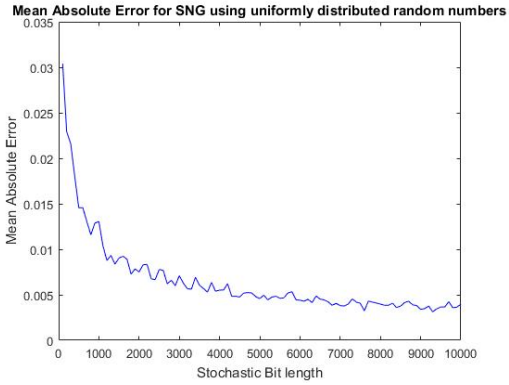

Figure 6.4: Mean absolute error for uniformly distributed random numbers

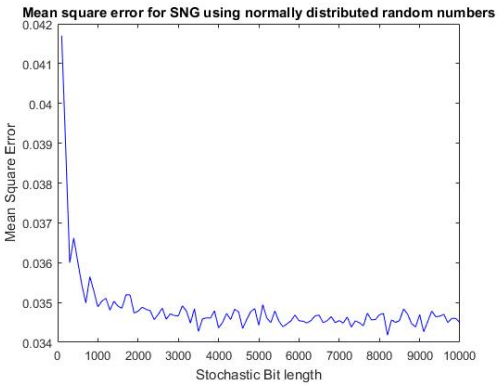

Figure 6.5: Mean square error for normally distributed random numbers

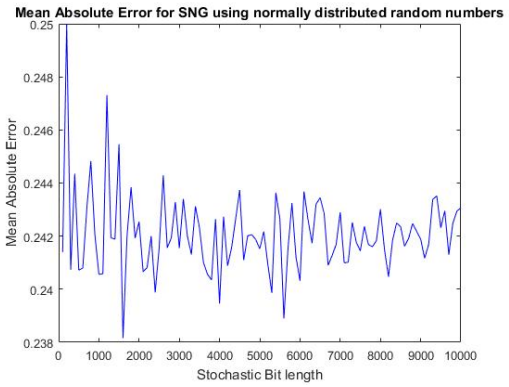

Figure 6.6: Mean absolute error for normally distributed random numbers

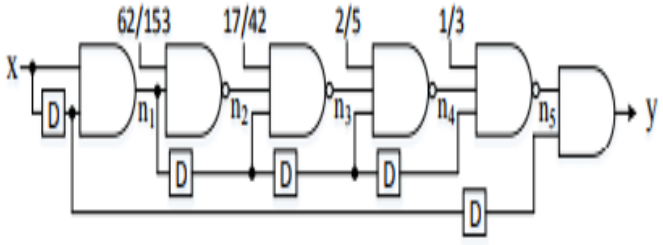

Figure 6.7: Stochastic tanh activation function circuit

6.2.1.3 Activation function

Two types of activation function were implemented for this network to arrive at the optimal one. Activation function is used in the hidden and output layer of the network. Tanh activation was implemented based on the 9th order McClarens series shown in equation 6.1 and is simplified to equation 6.2. This can be implemented for stochastic numbers using basic gates as shown in figure 6.3.

$$tanhax = ax - \frac{a^3x^3}{3} + 2a^5x^5 15 - \frac{17a^7x^7}{315} - \frac{62a^9x^9}{2835} \tag{6.1}$$

$$tanhax = ax(1 - \frac{a^2x^2}{3}(1 - 2a^2x^2 5(1 - \frac{17a^2x^2}{42}(1 - \frac{62a^2x^2}{153})))) \tag{6.2}$$

D flip flop is used to make a copy of the input bit stream and two same inputs

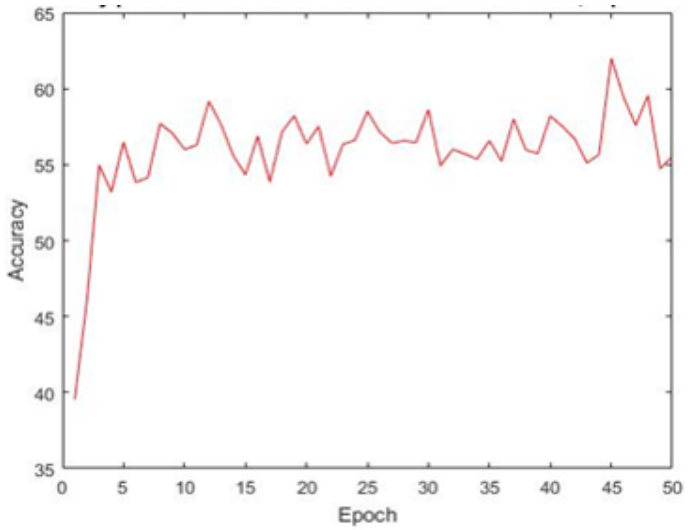

Figure 6.8: Accuracy obtained for stochastic tanh activation function

are given to the AND gate for multiplication and a NAND gate to achieve a negation of the product. This way the whole series can be generated to achieve tanh. Using tanh only 55% accuracy was obtained shown in Figure 6.4. This is due to the lack of precision and can be only improved by increasing the order of the series, which will intern increase the number of gates used and also increase the computation time.

sigmoid-like activation function were implemented for these networks, which is the count of 1's entering into the hidden node from all the inputs bitwise at a given time and then thresholding it with Half of the total number of inputs which is explained in detail in chapter 2.2.2. Use of this activation function gave above 85% accuracy for the network as shown in figure 6.6. From the results achieved, the sigmoid-like activation function was chosen for this implementation due to its less complex structure and high accuracy.

6.2.1.4 Learning rate

Learning rate is the parameter used for training the network by controlling the changes in bias and weight values. For small alpha values the network will train very slow and takes time to reach a good accuracy where as for larger Alpha values, the learning

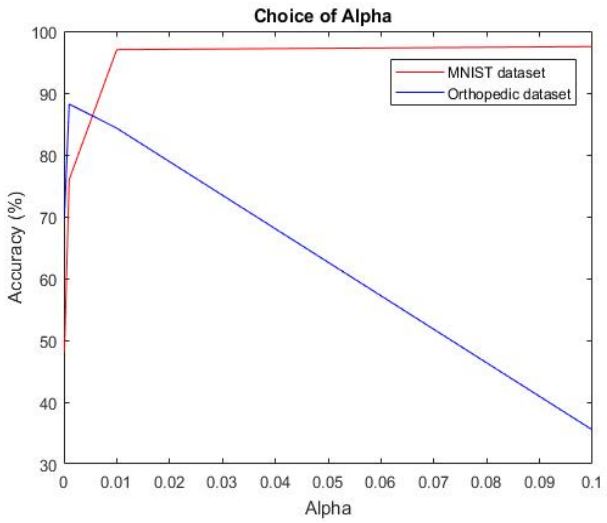

Figure 6.9: Stochastic ELM Accuracy Vs Alpha for MNIST and Orthopedic datasets

curve of the network will contain undulations and will not be exponential. Figure 6.3 shows the accuracy plots for different alpha values for MNIST and orthopedic datasets[31]. This shows that accuracy increases with increase in alpha value from 0.0001 to 0.1 and saturates after that in the case of the MNIST dataset. In the case of the orthopedic dataset a good accuracy is achieved below 0.001 because the network reaches good accuracy with 50 epochs. After 0.001, the acuuracy reduces because the network takes longer time to reach a good accuracy, thus unable to capture a good accuracy at 50 epochs.

6.2.2 Accuracy plots

The stochastic GPU implementation of the ELM network gives a test accuracy of 90.2% as shown in Figure 6.4 for MNIST dataset with alpha value of 0.001 and 512 hidden nodes. Maximum output accuracy obtained for orthopedic dataset is 87.5% with an alpha value of 0.001 and 512 hidden nodes as shown in Figure 6.11. There is a accuracy drop of around 3% compared to the normal implementation of these networks without stochastic logic. This is due to the random nature of stochastic logic, especially during the generation of stochastic number. One major contributor

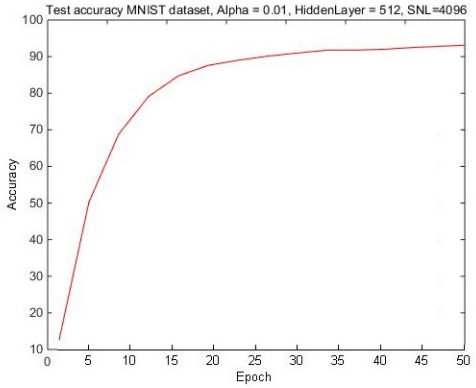

Figure 6.10: Stochastic ELM accuracy with MNIST dataset, Alpha =0.01

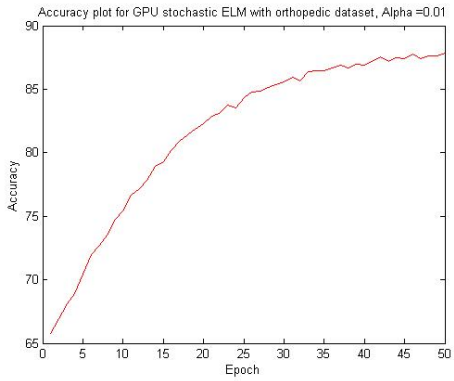

Figure 6.11: Stochastic ELM accuracy with Orthopedic dataset,Alpha =0.001

to the accuracy drop is the subtraction block in the training unit of the stochastic ELM, where only half of the sum of two inputs is obtained. The precision used for the floating point input numbers are 2^{16} for both the binary representation ANN and for the stochastic ANN before converting to stochastic. In some scenarios, where resource utilization is of prime concern, a 3% drop in accuracy might be tolerable.

6.2.3 Speedup for Stochastic ELM

The stochastic ELM is tested on two different GPUs. They are (i) GeForce GTX 480s Nvidia graphics card which has 480 cores, 2.0 CUDA capability and 1538 MB memory configuration and (ii) NVIDIA GeForce GTX 1050 Ti with 4GB memory with 768 cores and 6.1 CUDA capability. The timing results in Table 6.1 and 6.2 for Orthopedic dataset shows that the increase in computation time is relatively low compared to the increase in bit length, thus proving the GPU implementation works faster than CPU for higher bit streams. Among the two GPUs, the difference in speedup with the increase in bit stream decreases for the lower version of GPU and increases for the higher version of GPU, this is because GTX 480s with 1538 MB memory reaches the hardware limit for large bit streams much faster than the GTX 1050 Ti with 4 GB memory. Similar trend will follow for higher versions of GPU. With the better versions of CPUs, the speedup will relatively decrease but will still

SNL	CPU (mins)	GeForce GTX 480s (mins)	Speedup
2^{11}	1842.34	132.3	13.93
2^{12}	3661.42	204.2	17.93
2^{13}	7242.76	384	18.86

Table 6.1: Speedup of the Stochastic ELM network size 6X512X2 tested on orthopedic dataset over 50 epochs on GeForce GTX 480s GPU

SNL	CPU (mins)	GeForce GTX 1050 Ti (mins)	Speedup
2^{11}	1842.34	51.2	35.99
2^{12}	3661.42	60.4	60.61
2^{13}	7242.76	68.8	105.57

Table 6.2: Speedup of the Stochastic ELM network size 6X512X2 tested on orthopedic dataset over 50 epochs on GeForce GTX 1050 Ti GPU

be above one and will always be increasing with the increase in bit length due to the parallel operations in GPU.

6.3 Stochastic ESN in GPU

6.3.1 Ring Topology

The Ring Topology is a simple technique to connect the reservoir layer nodes compared to the random topology, since it has only one incoming connection from another reservoir layer node. Thus reducing the complexity of connection in reservoir layer and also reducing the calculation steps. Network size is 1 X 200 X 2 for EEG dataset (400 train and 100 test with 23 vector size) obtained a accuracy of 70.2% for 0.0001 alpha values and 73.5% for 0.001 alpha value. With greater alpha values the network reaches better accuracy earlier than the with lesser alpha values because the rate at which the network learns slows down for lower alpha values and will eventually reach lesser accuracies for the same number of epochs. The ring topology has achieved better accuracy than the random topology due to no random connections and simple architecture. There is around 15% drop in accuracy compared to a binary imple-

mentation of ESN which achieved 90% accuracy. This is due to the random nature of stochastic computation especially in the generation of stochastic numbers as it is compared with the random numbers. Also at the subtraction block for the generation of difference between the target and predicted output, there is a reduction in output by half. This due to the way the stochastic subtraction works as mentioned in chapter 2.2.3. Thus reducing the overall percentage of accuracy.

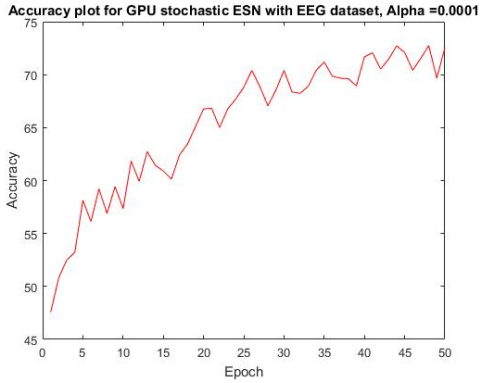

Figure 6.12: Stochastic ESN accuracy with EEG dataset for 200 hidden nodes and Alpha =0.0001

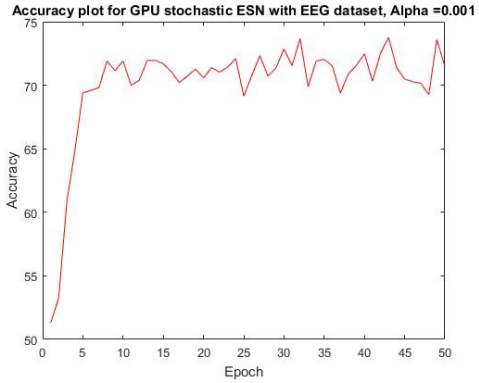

Figure 6.13: Stochastic ESN accuracy with EEG dataset for 200 hidden nodes and Alpha =0.001

6.3.1.1 Speedup for Stochastic ESN Ring topology

The stochastic ESN is tested on NVIDIA GeForce GTX 1050 Ti with 4GB memory with 768 cores and 6.1 CUDA capability. The timing results in Table 6.3 shows that the increase in computation time is relatively low compared to the increase in bit length in GPU where as in CPU, the computation time almost doubles with the increase in bit length, thus proving the GPU implementation works faster than CPU for higher bit streams. Also the speedup increases with the increase in bit length due to the number of operations in CPU increases with increase in bit stream length where as in GPU, The number of operations remain the same as the operations are performed in parallel. Though the time taken for each parallel operation in GPU increases with the increase in bit stream length, it is very less compared to the CPU

SNL	CPU (mins)	GeForce GTX 1050 Ti (mins)	Speedup
2^{11}	2227.958	85.833	32.75
2^{12}	5405.114	128.61	42.03
2^{13}	14379.78	209.16	68.75

Table 6.3: Speedup of the Stochastic ESN network with Ring topology, size 1 X 200 X 2 tested on EEG dataset over 50 epochs on GeForce GTX 1050 Ti GPU

SNL	CPU (mins)	GeForce GTX 1050 Ti (mins)	Speedup
2^{11}	4211.486	177.55	23.72
2^{12}	9898.15	287.93	34.37
2^{13}	18560.23	418.2	45.46

Table 6.4: Speedup of the Stochastic ESN network with Random topology, size 1X200X2 tested on EEG dataset over 50 epochs on GeForce GTX 1050 Ti GPU

operation.

6.3.2 Random topology

Percentage of connection is the parameter used in this network to denote the density of connection. There are different ways to connect Reservoir in RNN and it gives different results depending on the dataset. Stochastic ESN with Random topology is tested for EEG dataset for different percentage of connection as shown in figure. This method of connection is highly random and uncertain. There is no proper method or value at which a connection gives good accuracy. Only by a trial and error method the best percentage of connection for a particular dataset set can be determined. In this case GPU implementation of this network will help in rapid testing of the network to find the optimal solution. The accuracy obtained is 60.2% for 80% percent connection in reservoir layer. In general, the accuracy obtained is low for these types of reservoir connection due to the uncertainty in the connections.

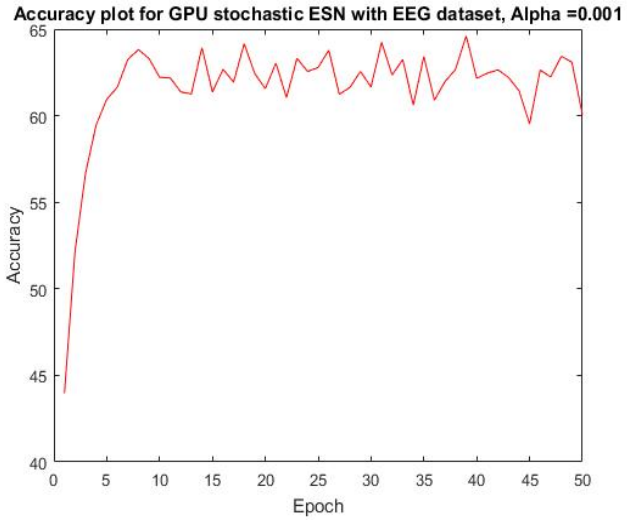

Figure 6.14: Stochastic ESN accuracy plot for Random connection topology, hidden layer nodes =200 and alpha =0.001

6.3.2.1 Speedup for Stochastic ESN random topology

The stochastic ESN random topology is also tested on NVIDIA GeForce GTX 1050 Ti with 4GB memory with 768 cores and 6.1 CUDA capability. The timing results in table 6.4 shows that the increase in computation time is relatively low compared to the increase in bit length in GPU where as in CPU, the computation time increases rapidly with the increase in bit length, thus proving that GPU implementation works better than CPU for higher bit stream lengths. Also the speedup increases with the increase in bit length but not as much as the stochastic ESN ring topology due to the random behavior of the reservoir layer and also due to the increase in the number of reservoir layer node connections, which ultimately increase the number of operations, where as in ring topology there are only one input to each reservoir node.

6.4 Discussion on precision

There is an accuracy drop of 3% between the binary representation ELM network in CPU and the stochastic ELM network in GPU. One of the reasons behind the

accuracy drop is the random nature of the stochastic computing. Apart from that, the precision of the CPU and GPU also plays a major role in the accuracy drop. GPU used in this work (Geforce GTX 1050 Ti) has a four 32-bit memory interface width where as the CPU (Intel i5-7360u) has a 64-bit instruction set width. The CPU calculates the results in double precision floating point numbers and The GPU calculates the results in single precision floating point numbers. The math libraries used for calculation varies for 32-bit and 64-bit mode compilation , thus resulting in the precision difference and ultimately the accuracy drop. Correctly rounded numbers in CPU slows down the execution speed compared to the GPU operation where an highly accurate result is not always needed in certain applications.

When converting the decimal number to stochastic, long bit lengths are required to achieve better accuracies. To convert a decimal number with 'n' decimal places require atleast 10^n bit length (for example: to represent 0.55 = 55/100 in stochastic logic with precision atleast 100 bits are required). In this work a double precision floating point number which uses 16 decimal digits were used which requires 10^16 stochastic number length to achieve the same precision when converting decimal to stochastic logic. Based on the discussion from chapter 6.2.1.1, 2^12 was chosen as the bit length for this work. Thus there is an accuracy drop in the proposed implementation due to the reduced bit length.

6.5 Hardware limitation

The plot for the increase in Stochastic bit length vs speedup was generated to determine the working of CPU and GPU based on computation time. Speedup is calculated by diving GPU computation time from CPU computation time. Figure 6.15 shows that there is a linear increase in speedup with the increase in bit stream length for the arithmetic blocks upto around 2^{12}, after which there is a drop in speedup. This is due to the increase in computation time of GPU is more compared to the increase in

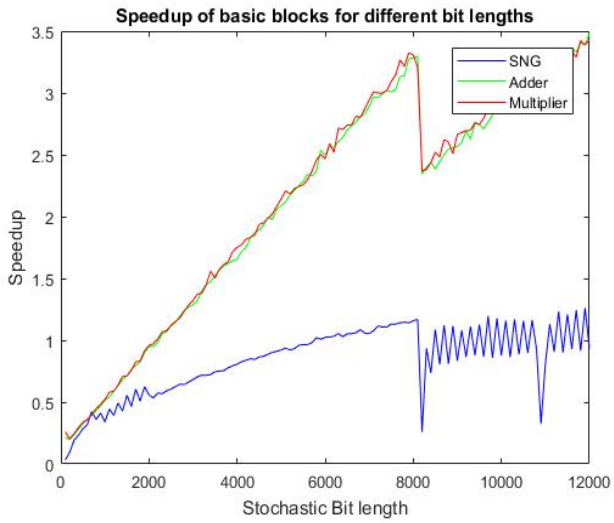

Figure 6.15: Stochastic bit length vs speedup

computation time of the previous step, where as the CPU computation time increases linearly thus showing a dip in the plot. This shows that the GPU Hardware limit is reached and it will future become slower with the increase in bit length. Similar trend has been shown in the stochastic number generator in the plot, except the speedup is comparatively lower due to more number of inter dependent bits compared to arithmetic units. This can be avoided by using higher version of GPUs like Geforce Titan, GTX Gefore 1080, etc.

Chapter 7

Conclusions

This work shows a GPU realization of a stochastic ELM and stochastic ESN for rapid prototyping. The design tested on various parameters like learning rate, bit length and number of hidden layers quickly narrows down on the best hardware realization. The design knobs have been identified for performance boost like stochastic number generator, activation function, reservoir layer connection and training unit. By using a hybrid stochastic-deterministic ELM and ESN design, the overall performance can be further improved. Non-complex arithmetic blocks like adders can be realized in the deterministic domain and the complex blocks can be realized in the non-deterministic domain.

The proposed stochastic ELM design was validated for two standardized datasets, an accuracy of 92.4% for MNIST dataset and 87.5% for orthopaedic dataset were observed. The proposed stochastic ESN was validated for EEG datasets and an accuracy of 73.5% was observed. Two types of activation functions namely tanh and sigmoidal were tested on these networks and narrowed down to sigmoidal for its simple design and better accuracy obtained compared to tanh activation function. The 2^{12} stochastic bit stream length was chosen by calculating the percentage of error with the increase in stochastic bit length. Reservoir layer is one of the design knobs where two different topologies - random and ring topologies were implemented to test the performance of the ESN network. Random topology had random connection

between the reservoir node based on the input percentage of connection, due to this randomness in the connections and complex architecture the accuracy obtained was low compared to ring topology.

A similar CPU model for both the networks were also implemented to calculate the performance boost thus proving the stochastic ELM in GPU works much faster compared to the CPU. Proposed stochastic ELM and stochastic ESN networks achieved a performance boost of 60.61X for Orthopedic dataset and 42.03x for EEG dataset with 2^{12} bit stream length when tested on a Nvidia GeForce 1050 Ti GPU with 4GB memory, 768 cores and 6.1 CUDA capability and Intel core i5-7360u, 4MB cache, 2 cores, 4 thread CPU. With the higher versions of GPU like Nvidia Titan XP, GeForce 1080, etc. better speedups can be achieved.

The increase in computation time was captured for increasing bit length to calculate the performance boost, which showed that the speedup increased with the increase in stochastic bit length, this is due to the large increase in computation time in CPU compared to GPU. This proved that large networks can be accelerated in GPUs to reduce the computation time significantly and also increase the accuracy of the network. For better versions of CPU, the speedup will accordingly decrease but will remain above one and will continue to follow the same pattern. Overall the speedup increases with increase in bit stream length, this is due to the lack of interdependencies between the bits.

Chapter 8

Future Work

The acceleration framework developed in this work can be implemented for other types of neural networks like multilayer perceptron, competitive learing and liquid state network with different types of activation functions like Relu, piece-wise linear, hyperbolic tangent, etc.

Hardware design for the proposed stochastic ELM and stochastic ESN can be implemented as the next step after software modelling. To verify the working of the hardware design, functional verification can be used. Function verification of any design requires golden input and output vectors for comparison. Proposed design is a bit exact software model, the input to this network and the output from this network can be captured and used as golden vectors for the functional verification of the Design Under Test (DUT). The intermediate input and output for the low level blocks and mid level blocks can also be captured to functionally verify each of these blocks as shown in Figure 8.1. Usually the hardware design development will be a top down approach, since the high level design is broken to mid level and low level blocks for designing and the verification will be a bottom up approach as each of the low level blocks are verified before the high level block.

The input golden vectors are captured from the output of the stochastic number generator and the output golden vectors of the network are captured at the output of the activation function at the output layer. These vectors are used to test the

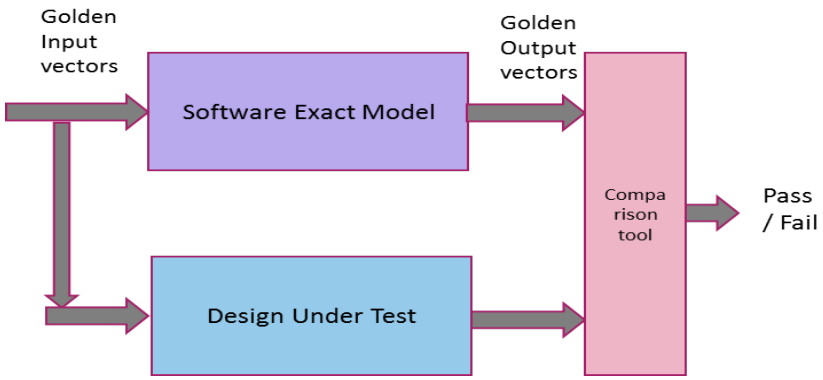

Figure 8.1: Functional verification block diagram

high level block. In order to verify the low level blocks, the input and output of the corresponding low level blocks are capture and compared with the output of the corresponding DUT by giving the same input. The comparator shown in the figure 8.1 compares the golden output vector against the output of the DUT to verify the block

Bibliography

[1] C. Magazine, "Machine learning using restricted boltzmann machines corpocrat magazine." [Online]. Available: corpocrat.com/2014/10/17/machine-learning-using-restricted-boltzmann-machines/.

[2] M. Written by Edgar G. Dawson, MD; Reviewed by Howard S. An, "Herniated discs: Definition, progression, and diagnosis," 2013. [Online]. Available: www.spineuniverse.com/conditions/herniated-disc/herniated-discs-definition-progression-diagnosis

[3] R. G. Andrzejak, K. Lehnertz, F. Mormann, C. Rieke, P. David, and C. E. Elger, "Indications of nonlinear deterministic and finite-dimensional structures in time series of brain electrical activity: Dependence on recording region and brain state," *Phys. Rev. E*, vol. 64, p. 061907, Nov 2001. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevE.64.061907

[4] E. M. Calimera, Andrea and M. Poncino.

[5] P. Ferreira, P. Ribeiro, A. Antunes, and F. M. Dias, *Artificial Neural Networks Processor – A Hardware Implementation Using a FPGA*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 1084–1086. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-30117-2_132

[6] G. Indiveri and S. C. Liu, "Memory and information processing in neuromorphic systems," *Proceedings of the IEEE*, vol. 103, no. 8, pp. 1379–1397, Aug 2015.

[7] E. Won, "A hardware implementation of artificial neural networks using field programmable gate arrays," *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, vol. 581, no. 3, pp. 816 – 820, 2007. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0168900207018530

[8] B. R. Gaines, *Stochastic Computing Systems*. Boston, MA: Springer US, 1969, pp. 37–172. [Online]. Available: http://dx.doi.org/10.1007/978-1-4899-5841-9_2

[9] A. Alaghi and J. P. Hayes, "Survey of stochastic computing," *ACM Trans. Embed. Comput. Syst.*, vol. 12, no. 2s, pp. 92:1–92:19, May 2013. [Online]. Available: http://doi.acm.org/10.1145/2465787.2465794

[10] J. von Neumann, *PROBABILISTIC LOGICS AND THE SYNTHESIS OF RELIABLE ORGANISMS FROM UNRELIABLE COMPONENTS*. Princeton University Press, 1956, pp. 43–98. [Online]. Available: http://www.jstor.org/stable/j.ctt1bgzb3s.5

[11] B. R. Gaines, "Stochastic computing," in *Proceedings of the April 18-20, 1967, Spring Joint Computer Conference*, ser. AFIPS '67 (Spring). New York, NY, USA: ACM, 1967, pp. 149–156. [Online]. Available: http://doi.acm.org/10.1145/1465482.1465505

[12] S. Shanmuganathan and S. Samarasinghe, *Artificial Neural Network Modelling*. Springer, 2016, vol. 628.

[13] R. Chauhan, H. Kaur, and M. Alam, "Article:data clustering method for discovering clusters in spatial cancer databases," *International Journal of Computer Applications*, vol. 10, no. 6, pp. 9–14, November 2010, published By Foundation of Computer Science.

[14] Q.-Y. Zhu, A. Qin, P. Suganthan, and G.-B. Huang, "Evolutionary extreme learning machine," *Pattern Recognition*, vol. 38, no. 10, pp. 1759 – 1763, 2005. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0031320305001809

[15] G.-B. Huang, D. H. Wang, and Y. Lan, "Extreme learning machines: a survey," *International Journal of Machine Learning and Cybernetics*, vol. 2, no. 2, pp. 107–122, 2011. [Online]. Available: http://dx.doi.org/10.1007/s13042-011-0019-y

[16] S. L. Bade and B. L. Hutchings, "Fpga-based stochastic neural networks-implementation," in *FPGAs for Custom Computing Machines, 1994. Proceedings. IEEE Workshop on*, Apr 1994, pp. 189–198.

[17] H. Jaeger, "The" echo state" approach to analysing and training recurrent neural networks-with an erratum note'," vol. 148, 01 2001.

[18] W. Maass, T. Natschlger, and H. Markram, "Real-time computing without stable states: A new framework for neural computation based on perturbations," *Neural Computation*, vol. 14, no. 11, pp. 2531–2560, 2002.

[19] Nvidia, "Nvidia launches the world's first graphics processing unit: Geforce 256," 1999.

[20] C. P. e. In: Tsutsui S., "Maitre o. (2013) understanding nvidia gpgpu hardware," *Massively Parallel Evolutionary Computation on GPGPUs. Natural Computing Series. Springer, Berlin, Heidelberg*.

[21] K. Yadav, A. Srivastava, and M. Ansari, "Parallel implementation of texture based medical image retrieval in compressed domain using cuda," *IJCA Special Issue on Novel Aspects of Digital Imaging Applications (DIA)*, no. 1, pp. 53–58, 2011, full text available.

[22] K.-S. Oh and K. Jung, "{GPU} implementation of neural networks," *Pattern Recognition*, vol. 37, no. 6, pp. 1311 – 1314, 2004. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0031320304000524

[23] R. Brito, S. Fong, K. Cho, W. Song, R. Wong, S. Mohammed, and J. Fiaidhi, "Towards implementation of residual-feedback gmdh neural network on parallel gpu memory guided by a regression curve," *The Journal of Supercomputing*, vol. 72, no. 10, pp. 3993–4020, 2016.

[24] D. Yudanov, "Gpu-based implementation of real-time system for spiking neural networks," 2009.

[25] "An American National Standard- IEEE Standard for Binary Floating-Point Arithmetic," IEEE and ANSI, Tech. Rep. Std 754-1985, 1985. [Online]. Available: http://mfkp.org/INRMM/article/1677576

[26] A. F.-F. Nathan Whitehead, "Precision performance: Floating point and ieee 754 compliance for nvidia gpus," 2011.

[27] B. K. Verma, S. B. Singh, and S. Akashe, "Ground bounce noise reduction aware combinational multi threshold cmos circuits for nanoscale cmos multiplier," *Frontiers of Optoelectronics*, vol. 6, no. 3, pp. 327–337, 2013.

[28] B. D. Brown and H. C. Card, "Stochastic neural computation. i. computational elements," *IEEE Transactions on Computers*, vol. 50, no. 9, pp. 891–905, Sep 2001.

[29] C. Merkel and D. Kudithipudi, "A stochastic learning algorithm for neuromemristive systems," in *2014 27th IEEE International System-on-Chip Conference (SOCC)*, Sept 2014, pp. 359–364.

[30] D. Kudithipudi, Q. Saleh, C. Merkel, J. Thesing, and B. Wysocki, "Design and analysis of a neuromemristive reservoir computing architecture for biosignal processing," *Frontiers in Neuroscience*, vol. 9, p. 502, 2016. [Online]. Available: https://www.frontiersin.org/article/10.3389/fnins.2015.00502

[31] M. Lichman, "UCI machine learning repository," 2013. [Online]. Available: http://archive.ics.uci.edu/ml

[32] M. P. Gerald L. Burke, "Backache: From occiput to coccyx," 2013. [Online]. Available: http://www.macdonaldpublishing.com/Chapter_3.html

[33] L.Deng, "The mnist database of handwritten digit images for machine learning research," *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 141–142, 2012.

[34] J. Christensen, M. Vestergaard, M. G. Pedersen, C. B. Pedersen, J. Olsen, and P. Sidenius, "Incidence and prevalence of epilepsy in denmark," *Epilepsy Research*, vol. 76, no. 1, pp. 60 – 65, 2007. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S092012110700188X

[35] S. Ramgopal, S. Thome-Souza, M. Jackson, N. E. Kadish, I. S. Fernndez, J. Klehm, W. Bosl, C. Reinsberger, S. Schachter, and T. Loddenkemper, "Seizure detection, seizure prediction, and closed-loop warning systems in

epilepsy," *Epilepsy Behavior*, vol. 37, no. Supplement C, pp. 291 – 307, 2014. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1525505014002297

[36] "Understanding seizures – the basics." [Online]. Available: www.webmd.com/epilepsy/understanding-seizures-basics